PB88-153671

Indoor Air Quality Modeling, Phase 2 Report
Residential Indoor Air Quality Simulation

National Bureau of Standards (NEL)
Gaithersburg, MD

Prepared for

Environmental Protection Agency, Washington, DC

Oct 87

U.S. Department of Commerce
National Technical Information Service
NTIS

PB88-153671

NBSIR 87-3661

Indoor Air Quality Modeling Phase II Report

James Axley

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Building Environment Division
Gaithersburg, MD 20899

October 1987

Prepared for:

U.S. Environmental Protection Agency
U.S. Department of Energy

REPRODUCED BY
U.S. DEPARTMENT OF COMMERCE
NATIONAL TECHNICAL
INFORMATION SERVICE
SPRINGFIELD, VA. 22161

NBSIR 87-3661

INDOOR AIR QUALITY MODELING
PHASE II REPORT

James Axley

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Building Technology
Building Environment Division
Gaithersburg, MD 20899

October 1987

Prepared for:
U.S. Environmental Protection Agency
U.S. Department of Energy

U.S. DEPARTMENT OF COMMERCE, C. William Verity, ***Acting Secretary***
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, ***Director***

ABSTRACT

This interim report presents the results of Phase II of the NBS General Indoor Air Pollution Concentration Model Project. It describes the theoretical basis of a general-purpose nonreactive contaminant dispersal analysis model for buildings, the computational implementation of a portion of this model in the program CONTAM86, and examples of the application of this model to practical problems of contaminant dispersal analysis. Presently the model is being extended to handle problems of reactive contaminant dispersal analysis and full computational implementation of all portions of the model is being completed.

The contaminant dispersal analysis model is based upon the idealization of building air flow systems as an assemblages of *flow elements* connected to discrete *system nodes* corresponding to well-mixed air zones within the building and its HVAC system. Equations governing the air flow processes in the building (e.g., infiltration, exfiltration, HVAC system flow, & zone-to-zone flow) and equations governing the contaminant dispersal due to this flow, accounting for contaminant generation or removal, are formulated by assembling element equations so that the fundamental requirement of conservation of mass is satisfied in each zone. The character and solution of the resulting equations is discussed and steady and dynamic solution methods outlined.

KEY WORDS: contaminant dispersal analysis , flow simulation, building simulation, building dynamics, computer simulation techniques, discrete analysis techniques,

ACKNOWLEDGEMENTS

Although the author of this report assumes full responsibility for the contents of the report it is important to acknowledge the contribution made by Richard Grot and George Walton who together with the author acted, in effect, as a project team.

Dr. Richard Grot of the Indoor Air Quality and Ventilation Group, Building Environment Division, National Bureau of Standards closely supervised all research reported in this document, providing essential critical evaluation and guiding the direction of the work by applying his considerable experience in the field and keen intellect to the task at hand. This he accomplished with his always engaging sense of humor and tireless enthusiasm.

The indoor air quality model presented in this report is based largely upon the work of George Walton of the Mechanical Systems and Controls Group, Building Environment Division, National Bureau of Standards. In fact, the present model should, properly, be presented as an extension of his earlier work. George was involved in Phase I and the early part of Phase II of this project and continued thereafter to provide his invaluable insight in the model development effort.

This work was supported by the Department of Energy, Contract IAG: DE-AI01-86CE21013, the Environmental Protection Agency, Contract IAG: DW-13931103-01-2, and the Department of Architecture, Cornell University.

CONTENTS

ABSTRACT ii
ACKNOWLEDGEMENTS iii
CONTENTS iv
PREFACE vii

PART I - Theoretical Basis

1. General Considerations 1-1
  1.1 Definition of Problem 1-2
  1.2 Modeling Approaches 1-4
  1.3 The Well-Mixed Macroscopic Model 1-6

2. Contaminant Dispersal Analysis 2-1
  2.1 Element Equations 2-1
  2.2 System Equations 2-3
  2.3 Boundary Conditions 2-5
  2.4 Elimination of Massless DOFs 2-7
  2.5 Qualitative Analysis of System Equations 2-8
  2.6 Solution of System Equations 2-17
    2.6.1 Steady State Behavior 2-18
    2.6.2 Free Response Behavior 2-18
    2.6.3 Dynamic Behavior 2-19

3. Air Flow Analysis 3-1
  3.1 Pressure Variation within Zones 3-1
  3.2 Element Equations 3-3
    3.2.1 Flow Resistance Element Equations 3-3
    3.2.2 Fan/Pump Element Equations 3-14
  3.3 System Equations 3-18
  3.4 Simple Examples 3-20
  3.5 Solution of Flow Equations 3-26
    3.5.1 Successive Substitution 3-26
    3.5.2 Newton-Raphson Iteration 3-30
    3.5.3 Incremental Formulation 3-33

4. Summary and Directions of Future Work 4-1

PART I References . . . . Ref I -1

PART II - CONTAM86 Users Manual

5. General Instructions 5-1

6. Command Conventions 6-1

7. Introductory Example 7-1

8. Command Reference 8-1
  8.1 Intrinsic Commands 8-1
    8.1.1 HELP 8-1
    8.1.2 ECHO 8-1
    8.1.3 LIST 8-1
    8.1.4 PRINT A=<array> 8-1
    8.1.5 DIAGRAM A=<array> 8-1
    8.1.6 SUBMIT F=<filename> 8-1
    8.1.7 RETURN 8-2
    8.1.8 QUIT 8-2
  8.2 CONTAM86 Commands 8-3
    8.2.1 FLOWSYS 8-3
    8.2.2 FLOWELEM 8-4
    8.2.3 STEADY 8-4
    8.2.4 TIMECONS 8-5
    8.2.5 Dynamic Analysis 8-7
      8.2.5.1 FLOWDAT 8-7
      8.2.5.2 EXCITDAT 8-9
      8.2.5.3 DYNAMIC 8-10
      8.2.5.4 Dynamic Analysis Example 8-11
    8.2.6 RESET 8-13

9. Example Problems 9-1
  9.1 Single Zone Examples 9-1
    9.1.1 Case 1: Contaminant Decay under
      Steady Flow Conditions 9-2
    9.1.2 Case 2: Contaminant Decay under
      Unsteady Flow Conditions 9-7
    9.1.3 Case 3: Contaminant Dispersal Analysis of an

Experimental Test 9-10
9.2 Two Zone Example 9-14
9.3 Full-Scale Multi-zone Residential Example 9-19

PART II References . . . . Ref II -1

Appendix- FORTRAN 77 Source Code

PREFACE

The work reported here is a product of the General Indoor Air Pollution Concentration Model Project initiated in 1985 at the National Bureau of Standards with the support of the U. S. Environmental Protection Agency and the U.S. Department of Energy. The fundamental objective of this project is to develop a comprehensive validated computer model to simulate dynamic pollutant movement and concentration variation in buildings. The scope of the project is ambitious; a full-scale, multi-zone building contaminant dispersal model that simulates flow processes (e.g., infiltration, dilution, & exfiltration) and contaminant generation, reaction, and removal processes is being developed.

During the planning stage of this project it was decided to organize efforts into three distinct phases:

Phase I: formulation of a general framework for the development of general indoor air quality analysis models (see [1] for report of Phase I work),

Phase II: development of a residential-scale model, based on the simplifying assumption that air is well-mixed within each building zone, providing simple simulation of HVAC system interaction, and

Phase III: extension of modeling capabilities to allow more complete simulation of HVAC system interaction and consideration of rooms that are not well-mixed.

This report presents a model that satisfies the scope and objectives set for Phase II of the "General Indoor Air Pollution Concentration Model" Project and, as such, completes Phase II efforts. The report is organized in two parts. In the first part of the report the theoretical basis of the model is presented;

Section 1: outlines the general aspects of indoor air quality simulation making the distinction between contaminant dispersal analysis and air flow analysis,

Section 2: presents the theoretical basis of contaminant dispersal analysis,

Section 3: presents the theoretical basis of air flow analysis.

The second part of the report presents the practical implementation of the contaminant dispersal analysis model in the program CONTAM86;

Sections 5 -8: provide a users manual for the program CONTAM86, and

Section 9: gives examples of application of CONTAM86, and its underlying theory, to problems of contaminant dispersal analysis.

The complete source code for CONTAM86 is listed in the appendix.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM.
BIBLIOGRAPHIC DATA SHEET (See instructions)

1. PUBLICATION OR REPORT NO.
NBSIR-87/3661

2. Performing Organ. Report No.
PB88 153671/AS

3. Publication Date

4. TITLE AND SUBTITLE
Indoor Air Quality Modeling - Phase II Report
Residential Indoor Air Quality Simulation

5. AUTHOR(S)
James W. Axley

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

U.S. Environmental Protection Agency
Washington, DC 20460

U.S. Department of Energy
1000 Independence Ave., SW
Washington, DC 20585

NBS Category
NBS- 230

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This interim report presents the results of Phase II of the NBS General Indoor Air Pollution Concentration Model Project. It describes the theoretical basis of a general-purpose nonreactive contaminant dispersal analysis model for buildings, the computational implementation of a portion of this model in the program CONTAM86, and examples of the application of this model to practical problems of contaminant dispersal analysis. Presently the model is being extended to handle problems of reactive contaminant dispersal analysis and full computational implementation of all portions of the model is being completed.

The contaminant dispersal analysis model is based upon the idealization of building air flow systems as an assemblages of flow elements connected to discrete system nodes corresponding to well-mixed air zones within the building and its HVAC system. Equations governing the air flow processes in the building (e.g., infiltration, exfiltration, HVAC system flow, and zone-to-zone flow) and equations governing the contaminant dispersal due to this flow, accounting for contaminant generation or removal, are formulated by assembling element equations so that the fundamental requirement of conservation of mass is satisfied in each zone. The character and solution of the resulting equations is discussed and steady and dynamic solution methods outlines.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

contaminant dispersal analysis; flow simulation; building simulation; building dynamics; computer simulation techniques; discrete analysis techniques.

13. AVAILABILITY

XXX Unlimited
☐ For Official Distribution, Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
XXX Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES
158

15. Price
MF
PC19.95/6.95

1. General Considerations

Airborne contaminants introduced into a building disperse throughout the building in a complex manner that depends on the nature of air movement in-to (infiltration), out-of (exfiltration), and within the building system, the influence of the heating ventilating and air conditioning (HVAC) systems on air movement, the possibility of removal, by filtration, or contribution, by generation, of contaminants by the HVAC system, and the possibility of chemical reaction or physical-chemical reaction (e.g., adsorption or absorption) of contaminants with each other or the materials of the buildings construction and furnishings.

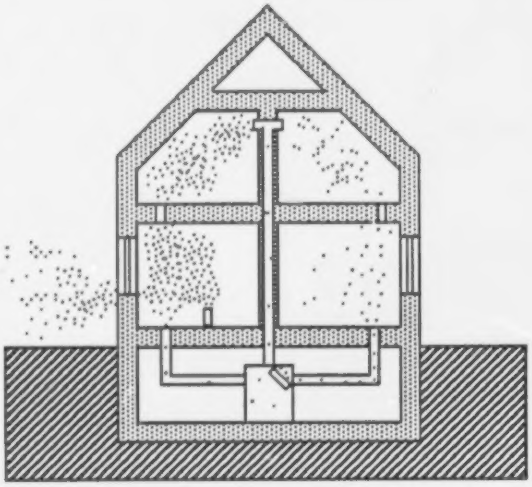


Fig. 1.1 Contaminant Dispersal in a Residence

Our immediate objective, here, is to develop a model of this dispersal process for residential-scale building systems that comprehensively accounts for all of these processes that affect the actual contaminant dispersal phenomena. We shall, however, attempt, to develop this residential-scale modeling capability within a more general context so that techniques developed here may be extended to more complex problems of indoor air quality analysis. To this end, in this section, the problem is given a general definition and the basic modeling strategy used to address this problem is outlined.

1.1 Definition of Problem

The building air flow system may be considered to be a three dimensional field within which we seek to completely describe the *state* of infinitesimal air parcels. The *state* of an air parcel will be defined by its temperature, pressure, velocity, and contaminant concentration (for each species of interest) - the *state variables* of the indoor air quality modeling problem.

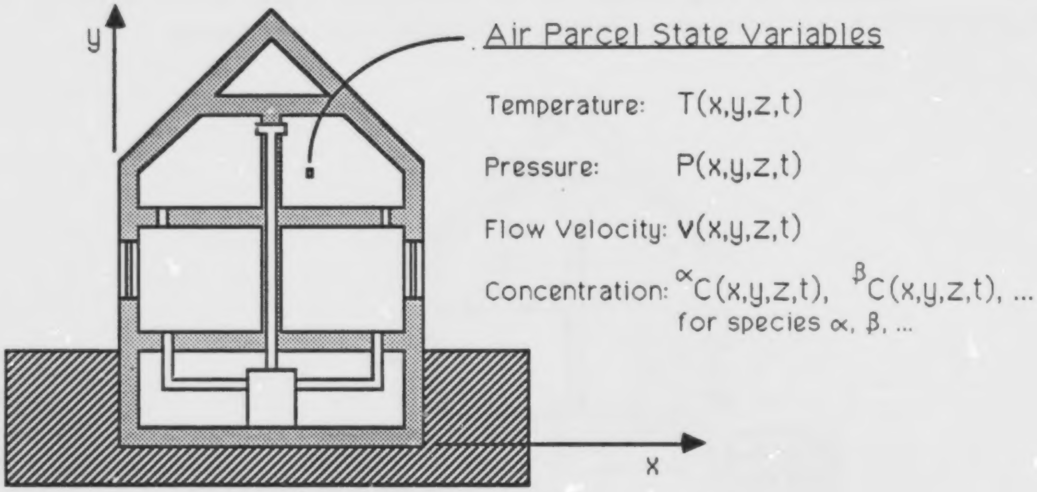

y
Air Parcel State Variables
Temperature: T(x,y,z,t)
Pressure: P(x,y,z,t)
Flow Velocity: V(x,y,z,t)
Concentration: ∝C(x,y,z,t), βC(x,y,z,t), ...
for species ∝, β, ...
x

Fig. 1.2 Air Parcel State Variables

Our immediate task is, then, to determine the spacial and temporal variation of the species concentrations within a building due to thermal, flow, and contaminant *excitation* driven by environmental conditions and the HVAC system and its control given building characteristics and their control. That is, we seek to determine;

${}^{\alpha}C(x,y,z,t)$; Contaminant "α" Concentration

${}^{\beta}C(x,y,z,t)$; Contaminant "β" Concentration

. . .

where;

C = species mass concentration or mass fraction

[=] mass of species/mass of air
α, β = species type indices
x, y, z = spacial coordinates
t = time

and shall refer to the process of determining the spacial and temporal variation of these species concentrations as *contaminant dispersal analysis* .

Contaminant dispersal analysis, for a single nonreactive species "α", depends on the air velocity field and its variation with time;

$$ {}^{\alpha}C(x,y,z,t) = {}^{\alpha}C(\,v(x,y,z,t)\,) \ \text{\& B.C.} \quad : \textit{Contam. Dispersal Anal.} \qquad (1.1) $$

But the air velocity field depends on the pressure field which is affected by the temperature field through buoyancy and, completing the circle, the temperature field is dependent on the velocity field;

$$ v(x,y,z,t) = v(\,P(x,y,z,t)\,) \ \text{\& B.C.} \quad : \textit{Flow Analysis} \qquad (1.2) $$

$$ P(x,y,z,t) = P(\,T(x,y,z,t)\,) \ \text{\& B.C.} \quad : \textit{Buoyancy Effects} \qquad (1.3) $$

$$ T(x,y,z,t) = T(\,v(x,y,z,t)\,) \ \text{\& B.C.} \quad : \textit{Thermal Analysis} \qquad (1.4) $$

where;
B.C = boundary conditions
v = air flow velocity
P = air pressure
T = air temperature

Thus, in general, contaminant dispersal analysis, for a single nonreactive species, is complicated by a *coupled nonlinear flow-thermal analysis* problem. Therefore, a comprehensive indoor air quality model will eventually have to address the related flow and thermal problems.

For cases of reactive contaminants, contaminant dispersal analysis, itself, will become a coupled (and, generally, nonlinear) analysis problem as individual species' concentrations will depend on other species' concentrations in addition to the air velocity field;

$${}^{\alpha}C(x,y,z,t) = {}^{\alpha}C(v, {}^{\beta}C, {}^{\gamma}C, \ldots) \quad : \textit{Species } \alpha \textit{ Dispersal Analysis} \qquad (1.5a)$$

$${}^{\beta}C(x,y,z,t) = {}^{\beta}C(v, {}^{\alpha}C, {}^{\gamma}C, \ldots) \quad : \textit{Species } \beta \textit{ Dispersal Analysis} \qquad (1.5b)$$

. . .

In this report we shall focus on single, nonreactive species dispersal analysis and the associated problem of flow analysis, for a completely defined thermal field and its variation. The approach taken, however, has been formulated to be compatible with thermal analysis modeling techniques developed earlier [2]. Presently, we are addressing the reactive, multiple species dispersal analysis problem and see no difficulty with extending the approach to this more complex situation.

1.2 Modeling Approaches

We shall attempt to solve the general field problems posed above by attempting to determine the state of air at discrete points in the building air flow system. It will be shown that this *spacial discretization* allows the formulation of systems of ordinary differential equations that describe the temporal variation of the state fields. Two basic approaches may be considered, one based upon the microscopic equations of motion (i.e., continuity, motion, and energy equations for fluids) and the other based upon a "well-mixed" zone simplification of macroscopic mass, momentum, and energy balances for flow systems (for a concise and complete review of these basic approaches see [3]).

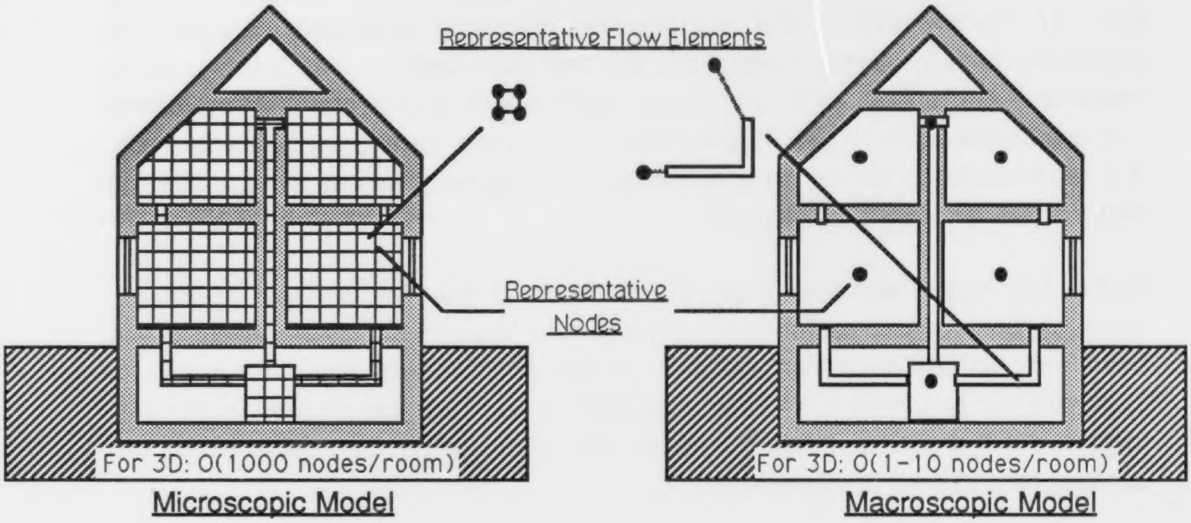

Representative Flow Elements
Representative Nodes
For 3D: O(1000 nodes/room)
Microscopic Model
For 3D: O(1-10 nodes/room)
Macroscopic Model

Fig. 1.3 Basic Spacial Discretization Approaches

In the microscopic modeling approach one of several techniques of the generalized finite element method, which includes the finite difference method [4], could be used to transform the systems of governing partial differential equations into systems of ordinary differential equations that then can be solved using a variety of numerical methods. The macroscopic modeling approach leads directly to similar systems of ordinary differential equations.

In both approaches the building air flow system is modeled as an assemblage of discrete flow *elements* connected at discrete system *nodes* . Systems of ordinary differential equations governing the behavior of elements are then formed and assembled to generate systems of ordinary differential equations that describe the behavior of the system as a whole (i.e., in terms of the spacial and temporal variation of the discrete state variables). These systems of equations may then be solved — given system excitation, initial conditions, and boundary conditions — to complete the analysis.

Virtually all computational procedures, except those used to form the element equations, would be practically identical for both approaches. From a practical point of view, however, microscopic modeling will involve on the order of 1000 nodes per room while the macroscopic model will involve on the order of only 10 nodes/room to realize acceptably accurate results. With six state variables for a single species - temperature, pressure, three velocity components and species concentration - the microscopic modeling approach can lead to extremely large systems of equations that therefore limit its use, at this time, to research inquiry. The macroscopic approach, resulting in systems of equations that are on the order of two magnitudes smaller than the microscopic approach, is a reasonable candidate for practical analysis, although it can not provide the detail of the microscopic approach.

Within this report we shall limit consideration to the macroscopic approach, although the specific techniques employed to implement this approach have been formulated to be compatible with the microscopic approach and it is expected that one may, in the future, be able to use both approaches in analysis to gain the benefits of detail in specific areas of the building system and yet account for full-system interaction.

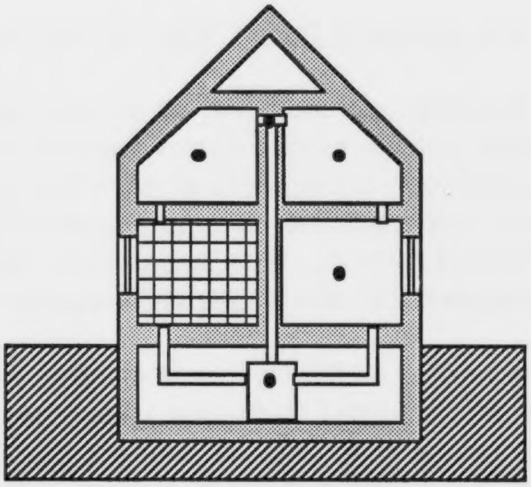
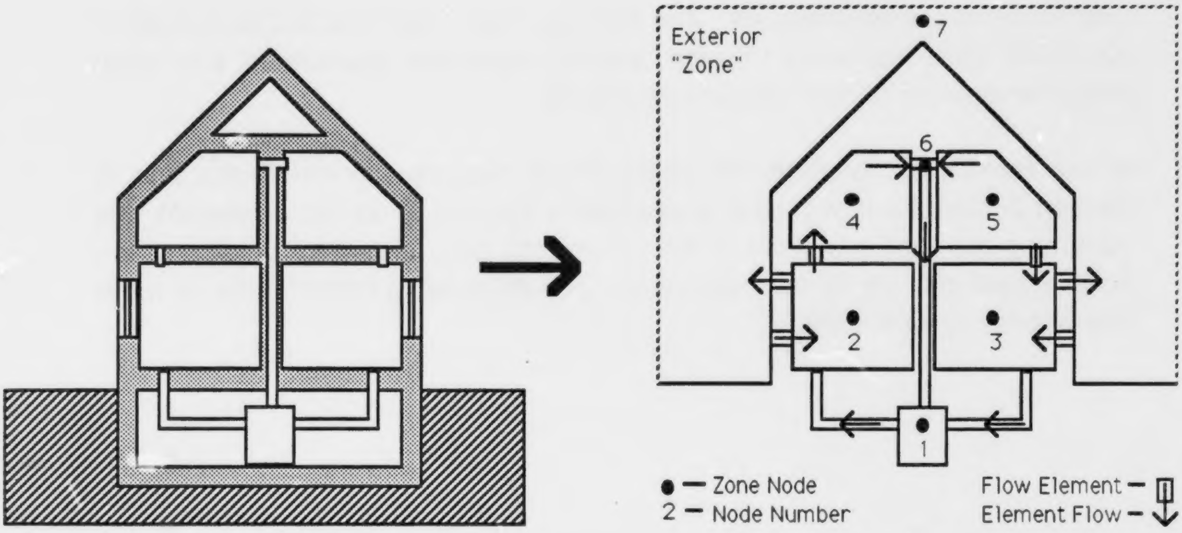


Fig. 1.4 Possible Hybrid Micro-Macro Discretization

1.3 The Well-Mixed Macroscopic Model

Here, the building air flow system shall be modeled as an assemblage of *flow elements* connected to discrete *system nodes* corresponding to well-mixed air *zones*.



Actual Building

Air-Flow System Idealization

Fig. 1.5 Well-Mixed Macroscopic Model

Limiting our attention to the contaminant dispersal and flow analysis problems we associate with each system node the discrete variables or *degrees of freedom* (DOFs) of pressure, air mass generation (typically zero), species concentration, species mass generation, and temperature;

$\{P\} = \{P_1, P_2, P_3, \ldots\}$: *Pressure DOFs* (1.6)

$\{W\} = \{W_1, W_2, W_3, \ldots\}$: *Air Mass Generation DOFs* (1.7)

$\{{}^{\alpha}C\} = \{{}^{\alpha}C_1, {}^{\alpha}C_2, {}^{\alpha}C_3, \ldots\}$: *Species α Conc. DOFs* (1.8)

$\{{}^{\alpha}G\} = \{{}^{\alpha}G_1, {}^{\alpha}G_2, {}^{\alpha}G_3, \ldots\}$: *Species α Gen. DOFs* (1.9)

$\{T\} = \{T_1, T_2, T_3, \ldots\}$: *Temp. DOFs* (1.10)

as well as the key system characteristic of nodal volumetric mass, V_1, V_2, V_3, ... The pressure, concentration, and temperature DOFs will approximate the corresponding values of the state field variables at the spacial locations of the system nodes.

With each element "e" in the system assemblage we note the *element connectivity* - the system nodes that the element connects - and identify an element air mass flow rate, w^e. The element mass flow rates will be related to the nodal state variables through specific properties associated with each particular element to form *element equations.*

In the formulation of both the contaminant dispersal model, presented in Section 2, and the flow model, presented in Section 3, we will *assemble* the governing element equations to form equations governing the behavior of the building system - the *system equations* - by demanding conservation of mass flow at each system node.

2. Contaminant Dispersal Analysis

In this section contaminant dispersal element equations are formulated. Demanding continuity of mass flow at each system node these element equations are then assembled to form contaminant dispersal equations governing the behavior of the full building system. Finally, methods for solution of the system equations are presented.

2.1 Element Equations

Two nodes[2-1] and a total mass flow rate, w^e, will be associated with each flow element, where flow from node i to j is defined to be positive. An element species concentration, ${}^{\alpha}C^e_k$, and an element species mass flow rate, ${}^{\alpha}w^e_k$, will be associated with each element node, k=i, j . The element species mass flow rate is defined so that flow from each node into the element is positive.

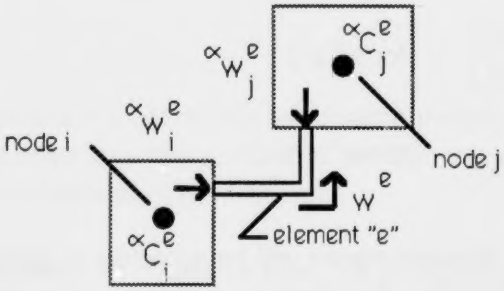


Fig. 2 .1 Contaminant Dispersal Element DOFs

It follows from fundamental considerations that these element variables are related directly to the element total mass flow rate as;

[2-1] The distinction between element nodes and systems nodes must be made because the element species concentration vector, $\{{}^{\alpha}\mathbf{C}^e\}$, is taken as a subset of the system species concentration vector, $\{{}^{\alpha}\mathbf{C}\}$.

$$\{{}^{\alpha}\mathbf{w}^{e}\} = |w^{e}|\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}\{{}^{\alpha}\mathbf{C}^{e}\} \quad ; \text{for } w^{e} \geq 0 \qquad (2.1a)$$

$$\{{}^{\alpha}\mathbf{w}^{e}\} = |w^{e}|\begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}\{{}^{\alpha}\mathbf{C}^{e}\} \quad ; \text{for } w^{e} \leq 0 \qquad (2.1b)$$

or

$$\boxed{\{{}^{\alpha}\mathbf{w}^{e}\} = [\mathbf{f}^{e}]\{{}^{\alpha}\mathbf{C}^{e}\}} \qquad (2.1c)$$

where;

$\{{}^{\alpha}\mathbf{w}^{e}\} = \{{}^{\alpha}w_i^{e}, {}^{\alpha}w_j^{e}\}^T$; element species mass flow rate vector

$\{{}^{\alpha}\mathbf{C}^{e}\} = \{{}^{\alpha}C_i^{e}, {}^{\alpha}C_j^{e}\}^T$; element species concentration vector

$[\mathbf{f}^{e}]$ = element total mass flow rate matrix

$$= |w^{e}|\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \quad ; \text{for } w^{e} \geq 0 \qquad (2.1d)$$

$$= |w^{e}|\begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix} \quad ; \text{for } w^{e} \leq 0 \qquad (2.1e)$$

For the purposes here, element nodes will be selected to correspond to specific system nodes, consequently, the element nodal species concentrations will have a one-to-one correspondence with the corresponding system node species concentrations.

If the element acts as a filter and removes a fraction, η , of the contaminant passing through the filter then the element flow rate matrix becomes;

$[\mathbf{f}^{e}]$ = element total mass flow rate matrix

$$= |w^{e}|\begin{bmatrix} 1 & 0 \\ (\eta-1) & 0 \end{bmatrix} ; \text{for } w^{e} \geq 0 \qquad (2.1f)$$

$$= |w^{e}|\begin{bmatrix} 0 & (\eta-1) \\ 0 & 1 \end{bmatrix} ; \text{for } w^{e} \leq 0 \qquad (2.1g)$$

The fraction, η , is commonly known as the "filter efficiency" and may have values in the range of 0.0 to 1.0.

2.2 System Equations

System equations that relate the system concentration DOFs, $\{{}^{\alpha}\mathbf{C}\}$, to the system generation DOFs,$\{{}^{\alpha}\mathbf{G}\}$, may be assembled from the element equations by first transforming the element equations to the system DOFs and then demanding conservation of species mass flow at each system node.

There exists a one-to-one correspondence between each element's concentration DOFs, $\{{}^{\alpha}\mathbf{C}^{e}\}$, and the system concentration DOFs, $\{{}^{\alpha}\mathbf{C}\}$, that may be defined by a simple *Boolean* transformation;

$$\{{}^{\alpha}\mathbf{C}^{e}\} = [{}^{\alpha}\mathbf{B}^{e}]\{{}^{\alpha}\mathbf{C}\} \tag{2.2}$$

where;

$[{}^{\alpha}\mathbf{B}^{e}]$ is an m x n Boolean transformation matrix consisting of zeros and ones; m = the number of element nodes (here, m=2); n = the number of system nodes

For example, an element with nodes i & j (or 1 & 2) connected to system nodes 5 & 9, respectively, of a 12-node system would have ones in the 1st row, 5th column and the 2nd row, 9th column and all other elements of the 2 x 12 Boolean transformation matrix would be set equal to zero.

In a similar manner, we may define a "system-sized vector" to represent the net species mass flow rate from the system node into an element "e", $\{{}^{\alpha}\mathbf{W}^{e}\}$, and relate it to the corresponding element species mass flow rate using the same transformation matrix, as;

$$\{{}^{\alpha}\mathbf{W}^{e}\} = [{}^{\alpha}\mathbf{B}^{e}]^{T}\{{}^{\alpha}\mathbf{w}^{e}\} \tag{2.3}$$

For an arbitrary system node n, with connected elements "a", "b", ... as indicated below in Fig. 2.2, we then demand conservation of species mass as;

$$\left\{ \sum_{\substack{\text{connected}\\ \text{elements}}} (\text{elem. species mass flow}) + \begin{pmatrix}\text{rate of change}\\ \text{of}\\ \text{species mass}\end{pmatrix} = \begin{pmatrix}\text{generation}\\ \text{of}\\ \text{species mass}\end{pmatrix} \right\}_{\text{system node n}} \tag{2.4}$$

or,

$$^{\alpha}W_n^a + {}^{\alpha}W_n^b + \ldots + V_n \frac{d\,^{\alpha}C_n}{dt} = {}^{\alpha}G_n \tag{2.5}$$

or, for the system as a whole;

$$\sum_{e = a,b,\ldots} \{^{\alpha}\mathbf{W}^e\} + [\mathbf{V}]\left\{ \frac{d\,^{\alpha}\mathbf{C}}{dt} \right\} = \{^{\alpha}\mathbf{G}\} \tag{2.6}$$

where;

$[\mathbf{V}]$ = diag(V_1, V_2, ...) ; the *system volumetric mass matrix*

V_i = the volumetric mass of node i

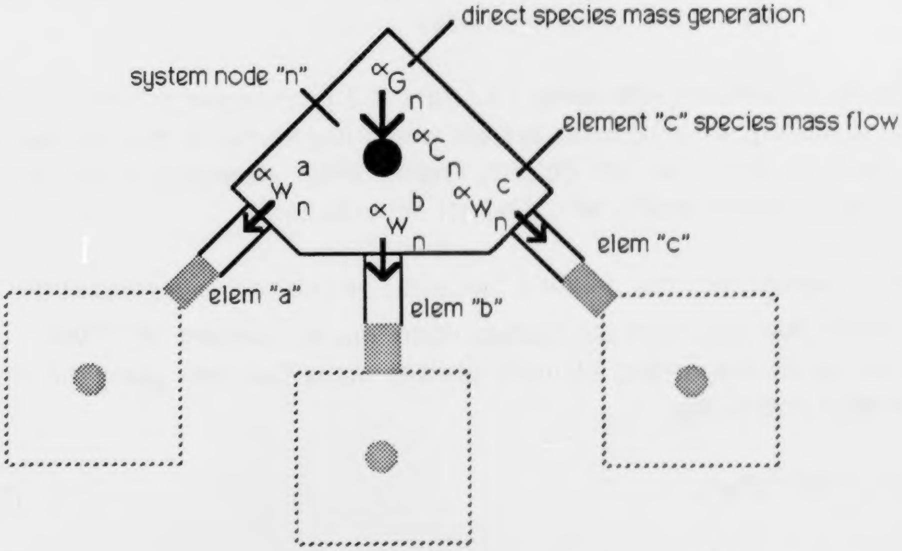

direct species mass generation
system node "n"
$^{\alpha}G_n$
$^{\alpha}C_n$
element "c" species mass flow
$^{\alpha}W_n^a$
$^{\alpha}W_n^b$
$^{\alpha}W_n^c$
elem "c"
elem "a"
elem "b"

Fig. 2.2 Conservation of Species α Mass Flow at System Node n

Substituting relations (2.2) and (2.3) we obtain the final result;

$$\boxed{[\mathbf{F}]\{{}^{\alpha}\mathbf{C}\} + [\mathbf{V}]\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\} = \{{}^{\alpha}\mathbf{G}\}} \qquad (2.7a)$$

where;

$$[\mathbf{F}] = \sum_{e=a,b,\dots} [{}^{\alpha}\mathbf{B}^{e}]^{T}[\mathbf{f}^{e}][{}^{\alpha}\mathbf{B}^{e}] \qquad (2.7b)$$

= the *system mass flow matrix*

$\equiv \mathsf{A}[\mathbf{f}^{e}]$; the direct assembly sum of element flow matrices

Equation (2.7a) defines the contaminant dispersal behavior of the system as a whole and is said to be *assembled* from the element equations through the relation given by equation (2.7b). The assembly process, as formally represented in equation (2.7b), has found widespread application in the simulation of systems governed by conservation principles and is, therefore, often represented by the so-called assembly operator A as indicated above. It should be noted that while the formal representation of the assembly process is important from a theoretical point of view it is generally far more efficient, computationally, to assemble the element equations directly, without explicitly transforming them (see, for example, the "LM Algorithm" in [24]).

2.3 Boundary Conditions

The variation of concentration or generation rate, but not both, may be specified at system nodes. Concentration or generation conditions in the discrete model are equivalent to boundary conditions in the corresponding continuum model and will, therefore, be referred to as such.

Formally then, we may distinguish between those DOFs for which concentration will be specified, $\{{}^{\alpha}\mathbf{C}_{c}\}$, and those for which generation rate will be specified, $\{{}^{\alpha}\mathbf{C}_{g}\}$, and partition the system of equations accordingly;

$$\begin{bmatrix} F_{cc} & F_{cg} \\ F_{gc} & F_{gg} \end{bmatrix} \begin{Bmatrix} {}^{\alpha}C_c \\ {}^{\alpha}C_g \end{Bmatrix} + \begin{bmatrix} V_{cc} & 0 \\ 0 & V_{gg} \end{bmatrix} \begin{Bmatrix} \dfrac{d^{\alpha}C_c}{dt} \\ \dfrac{d^{\alpha}C_g}{dt} \end{Bmatrix} = \begin{Bmatrix} {}^{\alpha}G_c \\ {}^{\alpha}G_g \end{Bmatrix} \qquad (2.8)$$

Using the second equation and simplifying we obtain;

$$[F_{gg}]\{{}^{\alpha}C_g\} + [V_{gg}]\left\{\frac{d^{\alpha}C_g}{dt}\right\} = \{{}^{\alpha}G_g\} - [F_{gc}]\{{}^{\alpha}C_c\} \qquad (2.9a)$$

or

$$\boxed{[\hat{F}]\{{}^{\alpha}\hat{C}\} + [\hat{V}]\left\{\frac{d^{\alpha}C}{dt}\right\} = \{{}^{\alpha}\hat{E}\}} \; . \qquad (2.9b)$$

where;

$[\hat{F}] \equiv [F_{gg}]$; the generation driven mass flow matrix

$\{{}^{\alpha}\hat{C}\} \equiv \{{}^{\alpha}C_g\}$; the generation driven nodal concentration vector

$\{{}^{\alpha}\hat{E}\} \equiv \{{}^{\alpha}G_g\} - [F_{gc}]\{{}^{\alpha}C_c\}$; the system *excitation* (2.9c)

It should be noted that the response of the system is driven by the system *excitation* involving both specified contaminant mass generation rates and contaminant concentrations which may, in general, vary with time.

Equation (2.9b), written in the standard form of a set of first order differential equations similar to the form of equation (2.7a), most directly defines the contaminant dispersal behavior of the system. The formation and solution of equation (2.9b) will be considered the central task of contaminant dispersal analysis.

The *response* of the system is defined by the solution of equation (2.9b) for the generation rate specified DOFs, $\{{}^{\alpha}C_g\}$. The generation rates, $\{{}^{\alpha}G_c\}$, required to maintain the specified concentrations, $\{{}^{\alpha}C_c\}$, may be determined from the response of the system to the specified excitation using the first equation of equation (2.8) as;

$$\{{}^{\alpha}\mathbf{G}_c\} = [\mathbf{F}_{cc}]\{{}^{\alpha}\mathbf{C}_c\} + [\mathbf{F}_{cg}]\{{}^{\alpha}\mathbf{C}_g\} + [\mathbf{V}_{cc}]\left\{\frac{d^{\alpha}\mathbf{C}_c}{dt}\right\} \tag{2.10}$$

Alternatively, one may numerically imposed specified concentration conditions by directly modifying equation (2.7a). The effect of an infinite source or sink, of the desired concentration, may be effected by scaling the appropriate diagonal terms of the system matrices by a large number and setting the corresponding generation rates equal to the product of the specified concentration and the scaled diagonal term. (The current version of CONTAM uses this strategy.)

2.4 Elimination of Massless DOFs

Often the analyst will define flow nodes within a complex building airflow system to model zones having negligibly small volumetric masses (e.g., junctions in HVAC system ductworks) and the analyst may prefer to model theses zones as if their nodal volumetric masses were zero. Additionally, the response at such nodes may be of little interest and the analyst may prefer to eliminate these nodal DOFs from consideration.

If the system of equations (2.9b) is partitioned into those DOFs having zero nodal volumetric masses, $\{{}^{\alpha}\mathbf{C}_Z\}$, and those having non-zero volumetric masses, $\{{}^{\alpha}\mathbf{C}_n\}$, as;

$$\begin{bmatrix} \hat{\mathbf{F}}_{zz} & \hat{\mathbf{F}}_{zn} \\ \hat{\mathbf{F}}_{nz} & \hat{\mathbf{F}}_{nn} \end{bmatrix} \begin{Bmatrix} {}^{\alpha}\hat{\mathbf{C}}_z \\ {}^{\alpha}\hat{\mathbf{C}}_n \end{Bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{V}}_{nn} \end{bmatrix} \begin{Bmatrix} \dfrac{d^{\alpha}\hat{\mathbf{C}}_z}{dt} \\ \dfrac{d^{\alpha}\hat{\mathbf{C}}_n}{dt} \end{Bmatrix} = \begin{Bmatrix} {}^{\alpha}\hat{\mathbf{E}}_z \\ {}^{\alpha}\hat{\mathbf{E}}_n \end{Bmatrix} \tag{2.11}$$

we may eliminate the massless DOFs from consideration by first solving for these DOFs using the upper equation;

$$\{{}^{\alpha}\hat{\mathbf{C}}_z\} = [\hat{\mathbf{F}}_{zz}]^{-1}\{\,\{{}^{\alpha}\hat{\mathbf{E}}_z\} - [\hat{\mathbf{F}}_{zn}]\{{}^{\alpha}\hat{\mathbf{C}}_n\}\,\} \tag{2.12}$$

and substituting this result in the lower equation tc obtain;

$$\boxed{[\tilde{F}]\{{}^{\alpha}\tilde{C}\} + [\tilde{V}]\left\{\frac{d^{\alpha}\tilde{C}}{dt}\right\} = \{{}^{\alpha}\tilde{E}\}} \tag{2.13a}$$

where;

$[\tilde{F}] \equiv [\hat{F}_{nz}][\hat{F}_{zz}]^{-1}[\hat{F}_{zn}]$; the reduced system flow matrix (2.13b)

$\{{}^{\alpha}\tilde{E}\} \equiv \{{}^{\alpha}\hat{E}_n\} - [\hat{F}_{zz}]^{-1}\{{}^{\alpha}\hat{E}_z\}$; the effective system excitation (2.13c)

$\{{}^{\alpha}\tilde{C}\} \equiv \{{}^{\alpha}\hat{C}_n\}$

$[\tilde{V}] \equiv [\hat{V}_{nn}]$

Equation (2.13a) is simply a reduced form of equation (2.9b); being a system of smaller size it may be solved more efficiently. In addition, the elimination of massless DOFs should help to avoid some numerical problems associated with round-off error. Eventhough the massless DOFs have been eliminated from consideration in equation (2.13a) their values may be recovered, at any time, using equation (2.12). (The current version of CONTAM does not eliminate massless DOFs.)

2.5 Qualitative Analysis of System Equations

It is important to keep in mind that we have developed equations that described the contaminant dispersal behavior of building idealizations, based upon assemblages of ideal flow elements, and have not, strictly speaking, developed equations that govern the behavior of the actual buildings being considered. Although it is hoped that these building idealizations will accurately describe the behavior of the actual buildings being modeled it is possible that they will not. In fact, it is quite possible to create idealizations that result in equations that have no solution, at all.

In this section, therefore, we shall consider the conditions that must be met to yield contaminant dispersal equations that have solutions and in so doing we shall also learn something about the general qualitative character of the solutions that are possible.

It should come as no surprise that building idealizations that satisfy conservation of total mass flow (i.e., as distinguished from species mass flow) will lead to system of equations that do, in fact, have solutions, but to get to this seemingly obvious conclusion we shall have to consider the details of the system flow and mass matrices and their impact upon the dynamic character of the system as a whole.

System Flow Matrix

The system flow matrix **[F]**, being a direct assembly sum of nonsymmetric element matrices, will also, in general, be nonsymmetric. The details of the assembly process reveal that the diagonal elements of the flow matrix are always positive and the off-diagonal elements negative. Furthermore, if the total mass flow into a system node is equal to the total mass flow out of a system node, then the diagonal elements of the flow matrix will be less than or equal to the "row sum" or the "column sum" of the corresponding off-diagonal elements.

More specifically, for a given system node i the diagonal element, F_{ii} , is simply equal to the total mass flow out of a node, the row sum of row i equals the sum of total mass flow into the node weighted by the filter efficiency factors (η - 1);

$$\text{row sum of row i} \equiv \sum_{\substack{j=1 \\ j \neq i}}^{n} |F_{ij}| = \text{weighted total mass flow into node i} \qquad (2.14)$$

and the column sum equals the sum of total mass flow out of the node weighted by the filter efficiency factors (η - 1);

$$\text{column sum of col. i} \equiv \sum_{\substack{j=1 \\ i \neq j}}^{n} |F_{ji}| = \text{weighted total mass flow out of node i} \qquad (2.15)$$

Therefore, if total mass flow is conserved at each node, we may assert;

$$F_{ii} \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |F_{ij}| \equiv \text{row sum of row i} \tag{2.16}$$

and

$$F_{ii} \geq \sum_{\substack{j=1 \\ i \neq j}}^{n} |F_{ji}| \equiv \text{column sum of col. i} \tag{2.17}$$

where the equality is strict when filter efficiencies of the elements connected to node i are zero (i.e., all $\eta = 0$) and the inequality holds if any of the connected outflow elements (for the row sum) or inflow elements (for the column sum) have nonzero filter efficiencies.

If all elements of a flow system idealization have nonzero filter efficiencies then the system flow matrix will be *strictly diagonally dominant* (i.e., for all i the inequalities above will hold); a condition that insures, by itself, the possibility of solution; that is to say, a sufficient condition to prove that the flow matrix would be *nonsingular.* For the (unlikely) limiting case where all elements have filter efficiencies equal to 1.0 the flow matrix becomes diagonal and, therefore, all zones act as independent (i.e.,uncoupled) single zone systems.

At the other (more likely) extreme where all elements have filter efficiencies equal to 0.0 the equalities of equations (2.16) and (2.17) hold for all nodes and the flow matrix is no longer strictly diagonally dominant and, therefore, may not be assumed to be nonsingular. We may show, however, that the important submatrix of the flow matrix identified earlier as the generation driven mass flow matrix is, in fact, nonsingular by demanding conservation of total mass flow of all subassemblages of system nodes and their inter-connecting elements and using some relatively esoteric theorems relating to the general class of matrices known as *M-matrices.*

An M-matrix may be defined in a number of alternative, but equivalent ways. Using the alternative employed by Funderlic and Plemmons [5] an M-matrix is a square nonzero real matrix with all off-diagonal elements nonpositive that has eigenvalues with nonnegative real parts. It may be shown [6] that a real square matrix [**A**], with positive diagonal elements and nonpositive off-diagonal elements;

a) is an M-matrix (possibly singular) if and only if it can be shown that $[\,[\mathbf{A}] + \xi[\mathbf{I}]\,]$ is a nonsingular M-matrix for all scalars $\xi > 0$ and

b) is a nonsingular M-matrix if [**A**] is strictly diagonally dominant

In the case at hand, clearly $[\,[\mathbf{F}] + \xi[\mathbf{I}]\,]$ is strictly diagonally dominant, and therefore a nonsingular M-matrix, for all scalars $\xi > 0$, (if, of course, total mass flow is conserved at all nodes). Thus we can conclude that [**F**] is an M-matrix, although it will be singular for the limiting case when all filter efficiencies are zero.

It has also been shown that each principal submatrix of an *irreducible* M-matrix (other than the M-matrix itself) is a nonsingular M-matrix [7]. The flow matrix would be said to be *reducible* if it is possible, using an appropriate numbering of the system nodes, to assemble the flow matrix in the form;

$$[\mathbf{F}] = \begin{bmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} \\ \mathbf{0} & \mathbf{F}_{22} \end{bmatrix} \tag{2.18}$$

where $\mathbf{F}_{11}$ and $\mathbf{F}_{22}$ are square matrices, otherwise [**F**] would be said to irreducible. Recalling thatsuperdiagonal term, F_{ij} ; $j > i$, corresponds to flow from node j to node i and a subdiagonal term, F_{ji} ; $j > i$, corresponds to flow from node i to node j, a flow matrix of the form of equation (2.18) would correspond to a flow system idealization having a total mass flow from subassembly 2 to subassembly 1, without a return flow from 1 to 2, and, therefore, conservation of total mass flow would be violated.

We may conclude, then, that;

a) the flow matrix, [**F**], will be an irreducible M-matrix and, therefore,

b) the generation driven mass flow matrix, $[\hat{\mathbf{F}}]$, a principal submatrix of the flow matrix will be a nonsingular M-matrix, if they are formed based upon a flow idealization that satisfies conservation of total mass flow

Inasmuch as the solution of the generation driven contaminant dispersal equations (equation (2.9b)) is the central task of contaminant dispersal analysis and the nonsingularity of the generation driven flow matrix is a necessary perequisite to assure the possibility of solution of these equations, the conclusion that the generation driven flow matrix will be nonsingular when the flow system idealization satisfies the condition of total mass conservation is of paramount importance. An additional property of nonsingular M-matrices provides the additional benefit of allowing efficient numerical solution strategies to be employed in the solution of these equations.

Nonsingular M-matrices, and therefore, properly formed $[\hat{\mathbf{F}}]$ matrices, have the important additional property that they may be factored into the product of lower, $[\mathbf{L}]$, and upper, $[\mathbf{U}]$, triangular matrices, $[\hat{\mathbf{F}}] = [\mathbf{L}][\mathbf{U}]$, by Gauss elimination without the need of pivoting in an efficient and numerically stable manner (i.e., resulting in no more accumulation of error that that which would result if pivoting were employed) [8]. Therefore, not only may we be certain that a properly formed flow matrix will lead to the possibility of solution but it will also allow the advantage of the use of very efficient methods of solution associated with *LU decomposition.*

System Volumetric Mass Matrix

By definition the system volumetric mass matrix, $[\mathbf{V}]$, is diagonal and nonnegative. In those instances when some nodal volumetric masses are so small that the analyst prefers to modeled them with zero values the system of contaminant dispersal equations may be reduced, by eliminating the massless equations (see section 2.4), to a form having an all positive, and therefore, nonsingular, volumetric mass matrix. The inversion of the positive volumetric mass matrix is trivial;

$$[\mathbf{V}]^{-1} = \mathrm{diag}(1/V_1, 1/V_2, \ldots 1/V_n) \quad ; V_i \neq 0 \qquad (2.19)$$

System Equations - Steady Flow

The generation driven contaminant dispersal equations, equation (2.9b), may now be rewritten in the form;

$$[\hat{\mathbf{V}}]^{-1}[\hat{\mathbf{F}}]\{{}^{\alpha}\hat{\mathbf{C}}\} + \left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\} = [\hat{\mathbf{V}}]^{-1}[{}^{\alpha}\hat{\mathbf{E}}] \qquad (2.20)$$

where, in general, the $[\hat{\mathbf{F}}]$ will vary with time.

The product matrix $[\hat{\mathbf{V}}]^{-1}[\hat{\mathbf{F}}]$ contains the essential dynamic character of the system being studied. For properly formed idealizations (being the product of a positive diagonal matrix and a nonsingular M-matrix [9]) it will be a nonsingular M-matrix and, therefore,

a) solutions to equation (2.20) will exist, and

b) the product matrix may also be factored into the product of lower, [**L**], and upper, [**U**], triangular matrices, $[\mathbf{V}]^{-1}[\hat{\mathbf{F}}] = [\mathbf{L}][\mathbf{U}]$, by Gauss elimination without the need of pivoting in an efficient and numerically stable manner .

We may gain some insight into the general character of solutions to equation (2.20) by considering the case of steady flow ($[\hat{\mathbf{F}}]$ constant) without excitation (i.e., the homogeneous case);

$$[\hat{\mathbf{V}}]^{-1}[\hat{\mathbf{F}}]\{{}^{\alpha}\hat{\mathbf{C}}\} + \left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\} = [0] \qquad (2.21)$$

Anticipating the result we try solutions of the form;

$$\{{}^{\alpha}\mathbf{C}\} = \{{}^{\alpha}\Phi\}e^{-t/\tau} \qquad (2.22)$$

where;

τ = decay time constant

$\{^{\alpha}\Phi\}$ = vector of unknown magnitudes

which, when substituted into equation (2.21) lead to the standard eigenvalue problem;

$$[\,[\mathbf{V}]^{-1}[\hat{\mathbf{F}}\,] - (1/\tau)[\mathbf{I}]\,]\,\{^{\alpha}\Phi\} = \{\mathbf{0}\} \tag{2.23}$$

The solution of this standard eigenvalue problem and its relation to the first order system of differential being considered is discussed elsewhere [10], [11] and is well beyond the scope of this report. Suffice it to say, for a properly formed flow system idealization of n nodes there will be n solutions to this eigenvalue problem consisting of n pairs of time constants, τ, (or equivalently their inverses, $1/\tau$ - the system eigenvalues) and their associated eigenvectors, $\{^{\alpha}\Phi\}$.

In some cases it may be possible to transform the product matrix $[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]$, by similarity transformations, to diagonal form leaving the eigenvalues on the diagonal as;

$$[\mathbf{S}]^{-1}[\,[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]\,]\,[\mathbf{S}] = \begin{bmatrix} (1/\tau_1) & 0 & \dots & 0 \\ 0 & (1/\tau_2) & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & (1/\tau_n) \end{bmatrix} \tag{2.24}$$

where;

[S] = the similarity transformation

For these cases it will be possible to express the general solution to the homogeneous problem, equation (2.21), as a linear combination of simple exponential decay terms;

$$\{^{\alpha}\mathbf{C}(t)\} = a_1\{^{\alpha}\,\Phi_1\,\}e^{-(t/\tau_1)} + a_2\{^{\alpha}\Phi_2\}e^{-(t/\tau_2)} \dots a_n\{^{\alpha}\Phi_n\}e^{-(t/\tau_n)} \tag{2.25}$$

where the scalar coefficients, a_1, a_2, ... a_n, are determined from the initial conditions using the similarity transformation employed as;

$$\begin{Bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{Bmatrix} = [\mathbf{S}]^{-1} \begin{Bmatrix} {}^{\alpha}C_1(t=0) \\ {}^{\alpha}C_2(t=0) \\ \dots \\ {}^{\alpha}C_n(t=0) \end{Bmatrix} \tag{2.26}$$

The n pairs of time constants and associated eigenvectors are often referred to as the system *modes* and the response of the system is often described in terms of the degree to which each mode participates. From the form of the free response, equation (2.25), it is clear that as time passes the contribution of those modes with larger time constants will dominate the character of the response until, eventually, the response, in all zones, will be dominated by the mode with the largest time constant and therefore will appear to be a simple exponential decay.

The similarity transformation [**S**] may be chosen as a matrix whose columns equal the eigenvectors, in this case, and, therefore, by equation (2.26) we can see that we may trigger a decay response in any single mode if we simply set the initial conditions equal to the corresponding eigenvector (or a scalar multiple of it), although, for some modes the eigenvectors will have negative components that, for contaminant dispersal problems, would not be physically admissible.

In general, the solution of the eigenvalue problem will be computationally demanding. However, for the limiting case discussed earlier, when all flow elements have filter efficiencies equal to 1.0, eigenanalysis is trivial. For this case the product matrix $[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]$ will be diagonal, therefore;

a) the time constants, τ_i, will be simply equal to (V_i/F_{ii}),

b) the similarity transformation will be equal to the identity matrix,

c) the eigenvectors will be equal to the unit vector corresponding to each DOF (i.e., the columns of the identity matrix), and

d) the scalar coefficients will equal the initial conditions corresponding to each DOF $\{a_1, a_2, \dots a_n\} = \{{}^{\alpha}C_1(t=0), {}^{\alpha}C_2(t=0), \dots {}^{\alpha}C_n(t=0)\}$.

For this limiting case all zones act independently as single zone "systems" and, therefore, these results follow directly from the more familiar single-zone theory.

For general contaminant dispersal systems we may apply the Gerschgorin Theorem [10], given the volumetric mass matrix is diagonal, to obtain a poorly bounded, but computationally inexpensive, estimate of the (real part of) system time constants as;

$$(1/\tau) = \frac{1}{V_i}\left(\hat{F}_{ii} \pm \sum_{j=1,2,\ldots}^{j\neq i} \hat{F}_{ij}\right) \quad ; \text{for all } i \tag{2.27}$$

This expression simplifies, exactly, to the values obtained for the limiting case discussed above, when all filter efficiencies equal 1.0, while at the other extreme, when all filter efficiencies are 0.0, it assures only that the system time constant will fall within the range;

$$\text{Min}\left(\frac{V_i}{2\hat{F}_{ii}}\right) \leq \tau \leq \infty \quad ; \text{ all filter efficiencies} = 0.0 \tag{2.28}$$

as, in these cases the off-diagonal row sum will be equal to the diagonal value of the flow matrix.

In some cases it will not be possible to diagonalize the product matrix $[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]$, but in these cases it will always be possible to transform the product matrix to a form known as the Jordan canonical form, an upper block-triangular matrix with the eigenvalues (inverse time constants) on the diagonal. For these cases, it will still be possible to express the general solution to the homogeneous problem, equation (2.21), as a combination of exponential decay terms, but now some of these decay terms will have factors equal to powers of time (i.e., in addition to terms like $e^{-(t/\tau)}$ we will have to include terms like $te^{-(t/\tau)}$, $t^2e^{-(t/\tau)}$, $t^3e^{-(t/\tau)}$, etc.).

In all cases the system time constants will have positive real parts, as the product matrix is a nonsingular M-matrix, and therefore all components making up the general solution will approach zero with time. That is to say, the homogeneous contaminant dispersal equations are *stable*; the concentration at all nodes will (eventually) approach zero. Furthermore, following the argument similar to that presented earlier in the discussion of the flow matrices, we may show that the sum of the product matrix and its transpose;

$$[\,[\mathbf{V}]^{-1}[\hat{\mathbf{F}}] + [[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]]^{T}\,]$$

is also a nonsingular M-matrix with positive (real parts of) eigenvalues and, therefore, the sum of the squares of the system concentrations (i.e., the *Euclidean* norm of the concentration vector) will decay at every instant of time [12];

$$\frac{d\|\{{}^{\alpha}\mathbf{C}(t)\|^2}{dt} < 0.0 \quad ; t \geq 0 \tag{2.29}$$

where;

$$\|\{{}^{\alpha}\mathbf{C}(t)\}\|^2 = (|{}^{\alpha}C_1(t)|^2 + |{}^{\alpha}C_2(t)|^2 + \ldots |{}^{\alpha}C_n(t)|^2)$$

These results are consistent with experience (and intuitive expectation) that while some nodal concentrations may at first increase with time (e.g., due to zone-to-zone mixing) in the long run all concentrations will diminish toward the zero level and at all times (some reasonable measure of) the mean concentration will also be diminishing.

The response of steady flow systems to nonzero excitation (i.e., the inhomogeneous case) may also be expressed in terms of linear combination of the eigenvectors of the product matrix $[\mathbf{V}]^{-1}[\hat{\mathbf{F}}]$. For practical contaminant dispersal analysis, however, it is more convenient to solve the system equations directly using numerical integration techniques that are not limited to steady flow cases.

2.6 Solution of System Equations

The governing system of equations, equation (2.9b), have the form of a system of first order linear differential equation with constant coefficients. In many practical situations, however, the mass flow rates will not be constant in time, and thus, in general, we may consider equation (2.9b) to be a system of first order differential equations with nonconstant coefficients. Here we shall consider the solution of these equations for;

1) Steady State: steady contaminant generation rates under conditions of steady element mass flow,

2) Free Response: transient decay of contaminant concentration under conditions of steady element mass flow,

3) Dynamic Response: to steady flow with unsteady generation rates, to unsteady flow with steady generation rates, or to unsteady flow with unsteady generation rates.

In the discussion below, equation (2.9b) will be written dropping the hat, ^, to simplify notation.

2.6.1 Steady State Behavior

For systems with steady element mass flows driven by steady contaminant generation rates and/or specified concentrations the response of the system will, eventually, come to a steady state (i.e., $\{d^{\alpha}\mathbf{C}/dt\} = 0$) given by the solution of;

$$[\mathbf{F}]\{^{\alpha}\mathbf{C}\} = \{^{\alpha}\mathbf{E}\} \tag{2.30}$$

As discussed in section 2.5 above this equation may be solved by LU decomposition without pivoting in an efficient and numerically stable manner.

2.6.2 Free Response Behavior

The free response behavior of steady flow systems has been discussed above and shown to be closely related to the solution of the eigenproblem given by equation (2.23) that yields system time constants and associated eigenvectors.

For steady flow systems knowledge of the system time constants provides invaluable insight into the dynamic character of the system yet eigenanalysis is computationally time consuming. It is, therefore, tempting to estimate the system time constants, after single-zone theory, by the ratio of the volumetric mass of each zone to the total air flow out of the zone. This estimate of system time constants will be designated as the *nominal system time constants* and, from the discussion in section 2.5, may be represented as;

$$\tau_i \approx \frac{V_i}{F_{ii}} \quad \text{; the } \textit{nominal system time constants} \tag{2.31}$$

For typical situations, however, the error bound on this estimate is very large (see section 2.5) and this estimate of the actual system time constants is likely to be a very poor estimate.

A variety of techniques exist that will provide better solutions to the governing eigenvalue problem and thereby provide better estimates of the actual system time constants [13]. The program CONTAM uses a relatively simple, published procedure, based on Jacobi iteration, that transforms the product matrix, $[\mathbf{V}]^{-1}[\mathbf{F}]$, to upper triangular form leaving the eigenvalues on the diagonal [14]. (The command TIMECONS in the program CONTAM reports both nominal and actual time constants for comparative purposes.)

2.6.3 Dynamic Behavior

The governing systems of equations, equation (2.9b), may be solved for cases of steady flow with general unsteady contaminant generation using any number of different finite difference solution schemes. Here we shall employ a general form predictor-corrector method.

For cases of unsteady flow it is likely that this same predictor-corrector solution scheme will prove useful, providing, of course, the system flow matrix, [**F**], is updated appropriately, although for cases of rapidly changing flow rates small time steps may be required to control error. If difficulties arise, an iterative scheme may have to be nested within the predictor-corrector time integration scheme.

A finite difference scheme for the approximate integration of the semidiscrete equation (2.9b) may be developed by dividing time domain into discrete steps;

$$t_{n+1} = t_n + \delta t \; ; \; n = 0,1,2,3 \ldots \tag{2.15}$$

t_0 = initial time

where;

δt = integration time step (often constant but may be variable)

demanding the satisfaction of equation (2.9b) at each of these steps;

$$[\mathbf{F}]\{^{\alpha}\mathbf{C}\}_{n+1} + [\mathbf{V}]\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} = \{^{\alpha}\mathbf{E}\}_{n+1} \tag{2.33}$$

where;

$\{^{\alpha}\mathbf{C}\}_{n+1} \equiv \{^{\alpha}\mathbf{C}(t_{n+1})\}$

$\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} \equiv \left\{\frac{d^{\alpha}\mathbf{C}(t_{n+1})}{dt}\right\}$

$\{^{\alpha}\mathbf{E}\}_{n+1} \equiv \{^{\alpha}\mathbf{E}(t_{n+1})\}$

Substituting into this equation the consistent difference approximation represented by;

$$\{^{\alpha}\mathbf{C}\}_{n+1} \approx \{^{\alpha}\mathbf{C}\}_n + (1-\theta)\delta t\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_n + \theta\delta t\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} \tag{2.34}$$

where;

$0 \le \theta \le 1$

$\theta = 0$ corresponds to the *Forward Difference* scheme

$\theta = 1/2$ corresponds to the *Crank-Nicholson* scheme

$\theta = 2/3$ corresponds to the *Galerkin* scheme

$\theta = 1$ corresponds to the *Backward Difference* scheme

a general implicit finite difference scheme is formulated;

$$[\,\theta\delta t[\mathbf{F}] + [\mathbf{V}]\,]\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} \approx \{^{\alpha}\mathbf{E}\}_{n+1} - [\mathbf{F}]\left\{\{^{\alpha}\mathbf{C}\}_n + (1+\theta)\delta t\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_n\right\} \tag{2.35a}$$

or, equivalently;

$$\left[[\mathbf{F}] + \left(\frac{1}{\theta\delta t}\right)[\mathbf{V}] \right]\{{}^{\alpha}\mathbf{C}\}_{n+1} \approx \{{}^{\alpha}\mathbf{E}\}_{n+1} + \left(\frac{1}{\theta\delta t}\right)[\mathbf{V}]\{{}^{\alpha}\mathbf{C}\}_{n} + (1-\theta)\delta t\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n} \quad (2.35b)$$

Computationally it is useful to implement this general finite difference scheme, equation (2.35), as a three step predictor-corrector algorithm;

$$\{{}^{\alpha}\tilde{\mathbf{C}}\}_{n+1} \equiv \{{}^{\alpha}\mathbf{C}\}_{n} + (1-\theta)\delta t\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n} \quad ;\ \textit{predictor} \quad (2.36a)$$

$$[\ (\theta\delta t)[\mathbf{F}] + [\mathbf{V}]\ ]\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} \approx \{{}^{\alpha}\mathbf{E}\}_{n+1} - [\mathbf{F}]\{{}^{\alpha}\tilde{\mathbf{C}}\}_{n} \quad ;\ \text{(i.e. eqn (2.35a))} \quad (2.36b)$$

$$\{{}^{\alpha}\mathbf{C}\}_{n+1} \approx \{{}^{\alpha}\tilde{\mathbf{C}}\}_{n+1} + (\theta\delta t)\left\{\frac{d^{\alpha}\mathbf{C}}{dt}\right\}_{n+1} \quad ;\ \textit{corrector} \quad (2.36c)$$

It should be noted that;

a) this algorithm is self-starting; given initial conditions, $\{{}^{\alpha}\mathbf{C}(t_0)\}$, equation (2.33) may be solved to obtain an estimate of the initial rate of change of nodal temperatures, $\{d^{\alpha}\mathbf{C}(t_0)/dt\}$, and the first predictor step, equation (2.36a) may then be computed, and

b) equation (2.36b) may also be solved by LU decomposition, without the need of pivoting; importantly then, the matrix $[\ (\theta\delta t)[\mathbf{F}] + [\mathbf{V}]\ ]$ may first be factored into the L and U product matrices and need not be refactored again until there is a change in the system flow matrix (i.e., due to unsteady element flows) and equation (2.36b) may then be solved, at minimum computational cost by back and forward substitution using the LU factors, for the first and each subsequent time step.

This predictor-corrector scheme has been analyzed by Taylor [15] and Huebner [16] and a more general predictor-multicorrector scheme that includes this *implicit* scheme has been analyzed by Hughes [17] for systems with constant coefficient matrices (i.e., [F] and [V] constant). For $\theta \geq 1/2$ this scheme leads to an unconditionally stable solution; $\theta \geq 3/4$ (approximately) leads to an unconditionally stable non-oscillatory solution; beyond this, Taylor makes some recommendations regarding selection of θ and step size, δt, to limit error while minimizing computational effort. (In the program CONTAM the default value of θ is set to 0.75, and may be reset by the user, and an estimate of the time step needed to limit error is reported (for the given initial conditions) using a method developed by Taylor [15].)

3. Air Flow Analysis

In this section air flow element equations are formulated that relate mass flow rate through flow elements to pressure differences across the elements, the assembly of these element equations to form equations governing the flow behavior of the building air flow system is discussed, and methods of solving these equations are presented. The formulation of the air flow equations presented herein is based, in large part, on the work of Walton [18], an example presented by Carnahan et. al. [19], and Chapter 33 of the ASHRAE Handbook 1985 Fundamentals [20].

3.1 Pressure Variation within Zones

A general model of building airflow systems, the "well-mixed macroscopic model", and system DOFs relating to this model were defined in Section 1.3 of this report. For this model, fluid density within any zone i, ρ_i, will be assumed constant and thus the variation of static pressure within a zone, $p_i(z)$, will be given by;

$$p_i(z) = P_i + \frac{g}{g_c} \rho_i (z_i - z) \tag{3.1}$$

where;

z_i = the elevation of node i relative to an arbitrary datum

z = elevation relative to an arbitrary datum

g = the acceleration due to gravity

g_c = dimensional constant (1.0 (kg m)/(N s^2))

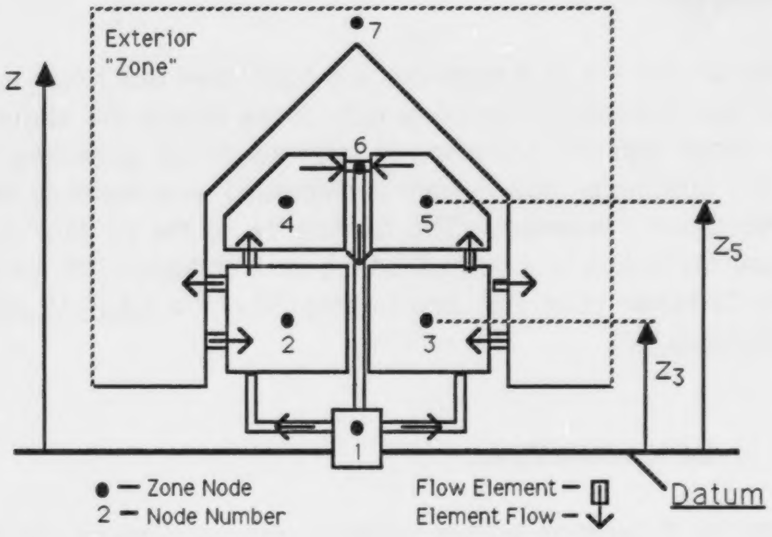


Fig. 3.1 Elevations Defined Relative to a Datum

Static pressures (i.e., under still conditions) acting on exterior surfaces may be approximated as;

$$p(z) = P_a - \frac{g}{g_c}\rho_a z \qquad \text{; on exterior surfaces, calm conditions} \qquad (3.2)$$

where P_a and ρ_a are the atmospheric pressure and air density at the level of the outdoor datum.

To account for pressures due to wind effects the pressure on any exterior surface may be approximated using published wind pressure coefficients [21] as;

$$p(z) = P_a + C_p \frac{\rho_a U_H^2}{2} \qquad \text{; on exterior surfaces, windy conditions} \qquad (3.3)$$

where C_p is a dimensionless pressure coefficient associated with the position on the exterior surface and the characteristics of the wind and U_H is the wind speed at the roof level of the building. Usually, local wind data will not be available; reference [21] suggests one modification of equation (3.3) to allow use of airport wind speed data.

(Strictly speaking equation (3.2) is exact for only a homogeneous atmosphere, i.e., of constant density. Typically, however, the lower atmosphere, at the scale of even the tallest buildings, has characteristics that fall between that of an isothermal atmosphere and a homogeneous atmosphere and equation (3.2) provides a very good estimate of air pressure for this range of conditions. Equation (3.3), on the other hand, provides only very approximate estimates of surface pressures. This is due to the great uncertainty of both pressure coefficients and the local wind speeds.)

3.2 Element Equations

Two classes of elements will be developed here; the first class, *flow resistance elements*, is a very general class that may be used to model a large variety of flow paths that provide passive resistance to flow (e.g., conduits, ducts, ductwork assemblies, small orifices such as cracks, etc.); the second class is developed to model fan-driven air flow. These two classes of elements should allow modeling of a large variety of complex and complete building airflow systems. It is anticipated, however, that special elements may need to be developed, in the future, to provide better models of some flow paths (e.g., flow through large openings such as doors and windows). Special elements may be developed using the resistance and fan/pump element formulations as examples of the general approach of element formulation.

3.2.1 Flow Resistance Element Equations

Resistance to flow will be modeled by flow elements having a single entry and exit (e.g., simple ducts, openings between zones, orifices, etc.). Flow components with multiple entries, exits, or both may be modeled as assemblages of these simpler elements.

Flow resistance elements shall be two-node elements. With each node we associate element pressure, P_i^e, temperature, T_i^e, and flow rate, w_i^e, DOFs (i.e., for flow from the node into the element). Element nodes are selected to have the same elevation as the zone nodes they connect[3-1].

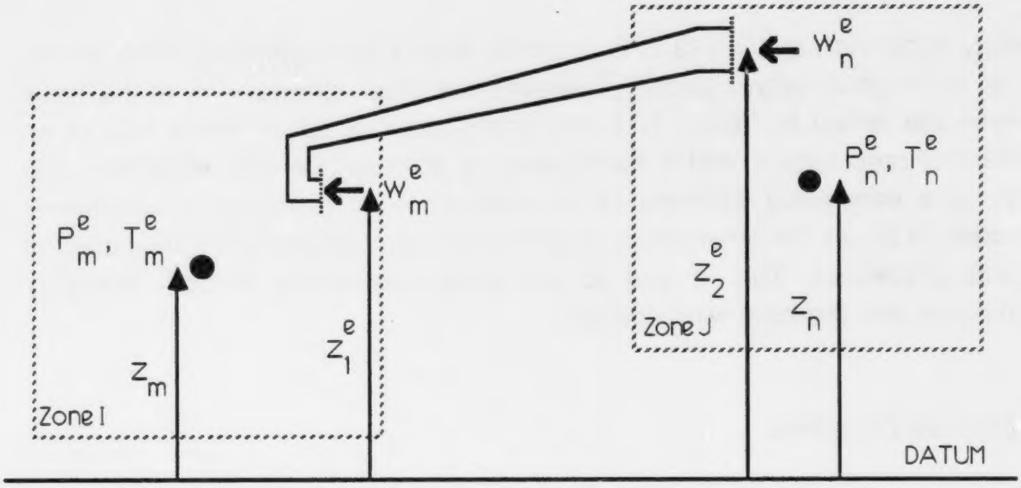


Fig. 3.2 Flow Resistance Element DOFs

Fluid flow within each flow resistance element is assumed to be incompressible, isothermal, and governed by the Bernoulli equation as applied to duct design [20];

$$\left(P_1 + \frac{\rho V_1^2}{2g_c}\right) - \left(P_2 + \frac{\rho V_2^2}{2g_c}\right) + \frac{g}{g_c}\rho(z^e_1 - z^e_2) = \sum \Delta p_o \qquad (3.4)$$

Where, for the purposes of developing the general element equations, the more conventional flow variables, indicated below, have been used;

P_1 , P_2 = entry and exit pressures, respectively
V_1 , V_2 = entry and exit mean velocities, respectively
g_c = dimensional constant, 1.0 (kg-m)/(N-sec^2)
g = the acceleration of gravity (e.g., 9.80665 m/sec^2)
ρ = density of fluid flowing through the element
z_1 , z_2 = elevations of entry and exits, respectively
w^e = mass flow rate through the element

[3-1] The distinction between element nodes and system nodes must be made because the element pressure vector, {$\mathbf{P}^e$}, is taken as a subset of the system pressure vector, {$\mathbf{P}$}.

$\sum \Delta p_o$ = the sum of all frictional and dynamic losses in the elements

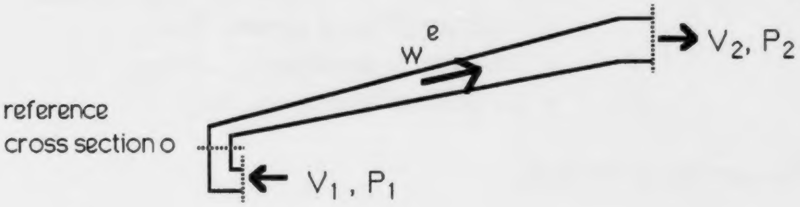


Fig. 3.3 Conventional Flow Variables

The losses, $\sum \Delta p_o$, are commonly related to the velocity pressure, $\rho V_o^2 / 2g_c$, of the fluid flow at reference cross sections "o";

$$\Delta p_o = C_o \frac{\rho V_o^2}{2g_c} \tag{3.5}$$

where; C_o = loss coefficient

- For conduits of constant cross-section:

$= f\,L/D_{eq}$

with;

f = dimensionless friction factor (see Chapter 33 equation (22) and/or Chapter 2 equations (16), (17), & (18) of ASHRAE 1985 Handbook of Fundamentals)
$\approx$ constant for turbulent flow (i.e. $Re > 2 \times 10^3$)
$\approx 64/Re$ for laminar flow (i.e. $Re < 2 \times 10^3$)

Re = $V_o D_{eq}/\mu$
μ = the fluid viscosity
L = length of conduit
D_{eq} = equivalent diameter of conduit
$= 4A/P_w$ = 4(flow area)/(wetted perimeter)

- For "fittings" of air flow systems see Appendix B, Chapter 33, ASHRAE Handbook 1985 Fundamentals.
- For flow through an orifice (Chapter 2, ASHRAE 1985 Fundamentals):

$$= \left(\frac{1}{C_d^2}\right)\left(\frac{D^4}{d^4} - 1\right)$$

C_d = orifice coefficient
≈ constant for turbulent flow (0.6 typically)
≈ (constant)/Re for laminar flow
D = diameter of approach to orifice
d = diameter of orifice opening

Thus the loss sum takes the form;

$$\sum \Delta p_o = \left(\frac{1}{2g_c}\right)\left(C_o \rho V_o^2 + C_p \rho V_p^2 + C_q \rho V_q^2 + \dots\right) \tag{3.6}$$

Recognizing that the mass flow rate, w^e, at each of these sections must be equal;

$$w^e = \rho V_1 A_1 = \dots = \rho V_o A_o = \rho V_p A_p = \rho V_q A_q = \dots = \rho V_2 A_2 \tag{3.7}$$

equation (3.6) may be rewritten in terms of mass flow rate as;

$$\sum \Delta p_o = (1/2g_c \rho)(C_o/A_o^2 + C_p/A_p^2 + C_q/A_q^2 + \dots)(w^e)^2 \tag{3.8}$$

and equation (3.4) then simplifies to;

$$(P_1 - P_2) + \frac{g\rho}{g_c}(z_1^e - z_2^e) = C^e (w^e)^2 \tag{3.9}$$

where;

$$C^e = (1/2g_c \rho)(-1/A_1^2 + \dots C_o/A_o^2 + C_p/A_p^2 + C_q/A_q^2 \dots +1/A_2^2) \tag{3.10}$$

Equation (3.9) may now be rewritten in terms of the element pressure DOFs, using equation (3.1), as;

$$(P^e_m - P^e_n) + \frac{g}{g_c}(\rho_m(z_m - z^e_1) + \rho(z^e_1 - z^e_2) + \rho_n(z^e_2 - z_n)) = C^e(w^e)^2 \qquad (3.11)$$

It may be seen from equation (3.11) that mass flow through element e is driven by the absolute pressure differences between zones ($P^e_m - P^e_n$) modified by buoyancy effects created by density differences that are, in turn, due to zone temperature differences.

Introducing a new variable, B^e, for the buoyancy induced pressure component;

$$B^e = \frac{g}{g_c}(\rho_m(z_m - z^e_1) + \rho(z^e_1 - z^e_2) + \rho_n(z^e_2 - z_n)) \qquad (3.12)$$

equation (3.11) may be rewritten as;

$$|w^e| = (C^e)^{-1/2}(|P^e_m - P^e_n + B^e|)^{1/2} \qquad (3.13a)$$

or

$$w^e = a^e(P^e_m - P^e_n) + a^e B^e \qquad (3.13b)$$

$$\text{where: } a^e = (C^e|P^e_m - P^e_n + B^e|)^{-1/2} \qquad (3.13c)$$

where the second form, equations (3.13b) and (3.13c), will provide the correct sign for w^e.

Variation of Flow With Zone Pressure

It is useful, at this point, to develop analytical expressions for the variation of mass flow with zone pressure. This expressions will be seen to be useful for solving the nonlinear flow system equations using schemes based upon the classical Newton-Raphson iteration method. Therefore, from equations (3.13b) and (3.13c) we obtain;

$$\frac{\partial w^e}{\partial P^e_m} = -\frac{1}{2}(C^e)^{-3/2}\frac{\partial C^e}{\partial P^e_m}(|P^e_m - P^e_n + B^e|)^{1/2} + \frac{1}{2}(C^e)^{-1/2}(|P^e_m - P^e_n + B^e|)^{-1/2} \qquad (3.14a)$$

$$\frac{\partial w^e}{\partial P^e_n} = -\frac{1}{2}(C^e)^{-3/2}\frac{\partial C^e}{\partial P^e_n}(|P^e_m - P^e_n + B^e|)^{1/2} - \frac{1}{2}(C^e)^{-1/2}(|P^e_m - P^e_n + B^e|)^{-1/2} \qquad (3.14b)$$

and from equation (3.10) we obtain;

$$\frac{\partial C^e}{\partial P^e_m} = (1/2g_c\rho)(A_o^{-2}\frac{\partial C_o}{\partial P^e_m} + A_p^{-2}\frac{\partial C_p}{\partial P^e_m} + A_q^{-2}\frac{\partial C_q}{\partial P^e_m} + \ldots) \qquad (3.15a)$$

$$\frac{\partial C^e}{\partial P^e_n} = (1/2g_c\rho)(A_o^{-2}\frac{\partial C_o}{\partial P^e_n} + A_p^{-2}\frac{\partial C_p}{\partial P^e_n} + A_q^{-2}\frac{\partial C_q}{\partial P^e_n} + \ldots) \qquad (3.15b)$$

that is, the variation of C^e with pressure is simply a weighted sum of the variation of individual pressure loss coefficients contributing to the total pressure loss along the element. Analytical expressions for these partial derivatives of the pressure loss coefficients are not easily formulated, but by considering limiting cases of flow we can gain some insight.

In general, the loss coefficients depend, in a rather complex and poorly understood way, upon the nature of flow, as indicated by the Reynolds number, Re, and detailed characteristics of the flow geometry (e.g., roughness, constrictions, etc.). For many situations, however, the loss coefficients are practically constant for the limiting case of fully turbulent flow (i.e., $Re > 10^6$), at one extreme, and proportional to 1/Re for laminar flow (i.e., $Re < 2 \times 10^3$) at the other;

$$C_o \approx \text{constant} \qquad (3.16)$$

for fully developed turbulent flow

$$C_o \approx C^*_o/Re = C^*_o\,\mu/\rho D_o V_o \qquad (3.17)$$

fully developed laminar flow

where; C^*_o = constant
- For conduits of constant cross-section;
  $= 64\ L/D_{eq}$
- For "fittings" values of C^*_o are not available; it may be reasonable to estimate values based upon equivalent lengths of conduits used in turbulent flow calculations (e.g. see ASHRAE 1985 Handbook of Fundamentals Chptr 34).
- For flow through an orifice;
  = ??

μ = fluid viscosity
D_o = a characteristic dimension of the flow geometry

In fully developed turbulent flow, with each of the pressure loss coefficients constant, the partial derivatives of equations (3.15) become zero and consequently the first term of equations (3.14) becomes zero and, using equations (3.13), may be simplified to;

$$\frac{\partial w^e}{\partial P^e_m} = \frac{1}{2} a^e \quad \text{; for fully turbulent flow} \tag{3.18a}$$

$$\frac{\partial w^e}{\partial P^e_n} = -\frac{1}{2} a^e \quad \text{; for fully turbulent flow} \tag{3.18b}$$

Limiting consideration to flow resistance elements of constant cross-section, we may formulate a modified expression for laminar flow in an element, in a manner similar to that used to formulate equations (3.13). We obtain;

$$w^e \approx a^e_L (P^e_m - P^e_n) + a^e_L B^e \tag{3.19a}$$

$$\text{where: } a^e_L = (2g_c \rho/\mu)\left(\frac{C^*_o}{D_o A_o} + \frac{C^*_p}{D_p A_p} + \frac{C^*_q}{D_q A_q} + \dots \right) \tag{3.19b}$$

for which the evaluation of the variation of flow with pressure is straightforward;

$$\frac{\partial w^e}{\partial P^e_m} = a^e_L \quad \text{; laminar flow, constant cross section} \qquad (3.20a)$$

$$\frac{\partial w^e}{\partial P^e_n} = - a^e_L \quad \text{; laminar flow, constant cross section} \qquad (3.20b)$$

It is instructive to compare the fully turbulent flow equation, equation (3.13) with C^e constant, with this particular case (i.e., constant cross section) fully laminar flow equation;

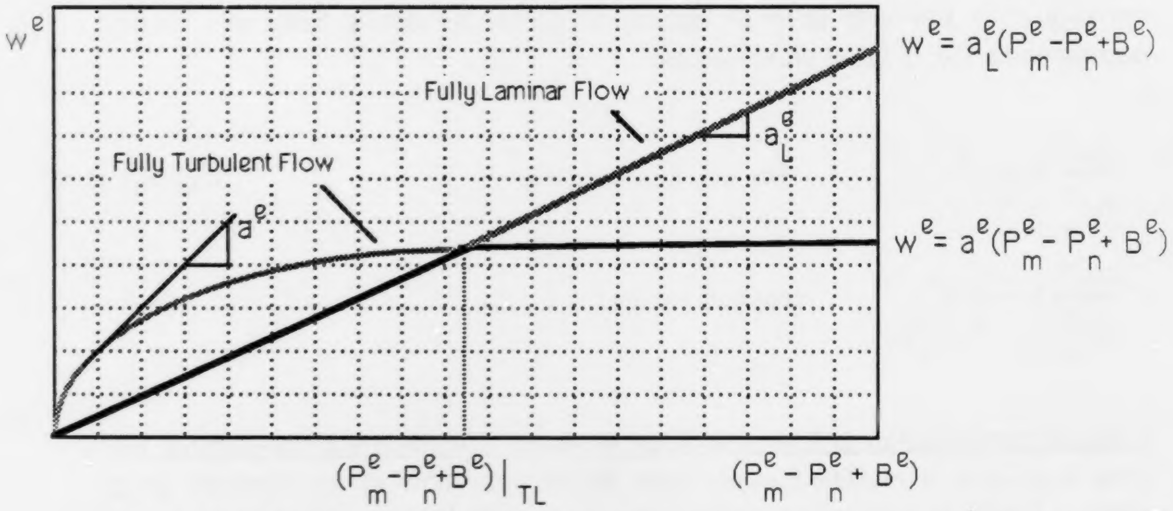


Fig. 3.4 Limiting Case Flow Relations- Elements of Constant Cross-Section

It is seen that a^e, the tangent slope of the fully turbulent curve, becomes unbounded as flow approaches zero-flow conditions while a^e_L does not.

If the variations of the pressure loss coefficients, C_o , C_p , C_q , ... , with flow are well defined (i.e., for conduits: if the friction factor relations are reliable) then the flow defined by equations (3.13) should asymptotically approach these two curves at the upper and lower limits of flow. (Note: this is not to say that these two curves provide an upper or lower bound to flow magnitude, in fact, they do not (e.g., orifice flow: see reference [22] Fig. 18)).

Our purpose, here, is not to use these limiting-case flow relations in place of the more general relation of equations (3.13), but rather to use these limiting cases to provide an estimate of the variation of element flow with zone pressure to be used in nonlinear solution algorithms. Specifically, we shall only employ equations (3.19) and (3.20) for very low flow conditions, when the more general expression for flow, equation (3.13b), and the approximation for the variation of flow with pressure, equations (3.18), will tend to become unbounded.

Matrix Formulation of the Element Flow Equations

The element equations may be recast into matrix form, using the element DOFs defined above, by first noting;

$$w^e = w_m^e = -w_n^e \tag{3.21}$$

thus;

$$\boxed{\{w_{net}^e\} = [a^e]\{P^e\} + \{w_B^e\}} \tag{3.22a}$$

where;

$$\{w_{net}^e\} = \{w_m^e, w_n^e\}^T \tag{3.22b}$$

= the element net mass flow rate vector

$$\{P^e\} = \{P_m^e, P_n^e\}^T \tag{3.22c}$$

= the element pressure vector

$$[a^e] = a^e \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{; for all but very low flow conditions} \qquad (3.22d)$$

$$= a_L^e \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{; for very low flow conditions} \qquad (3.22e)$$

= matrix of pressure-flow coefficients

$$\{w_B^e\} = a^e B^e \{\, 1 \;\; -1 \,\}^T \quad \text{; for all but very low flow conditions} \qquad (3.22f)$$

$$= a_L^e B^e \{\, 1 \;\; -1 \,\}^T \quad \text{; for very low flow conditions} \qquad (3.22g)$$

= bouyancy-induced mass flow rate vector

and;

$$\frac{\partial\{w_{net}^e\}}{\partial\{P^e\}} = \frac{a^e}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{; for all but very low flow conditions} \qquad (3.23a)$$

$$= a_L^e \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{; for very low flow conditions} \qquad (3.23b)$$

The element pressure-flow coefficients a^e and a_L^e are defined in such a way that they are always positive, therefore, the matrix of pressure-flow coefficients will be positive semi-definite.

Some complicating details deserve special note;

a) the direction of flow will be determined by the sign of $(P_m^e - P_n^e + B^e)$; if positive, the flow will be from m to n,

b) the density ρ, of the fluid flowing through the element, will depend on the direction of flow;

$\rho = \rho_m$; for flow from m to n

$\rho = \rho_n$; for flow from n to m

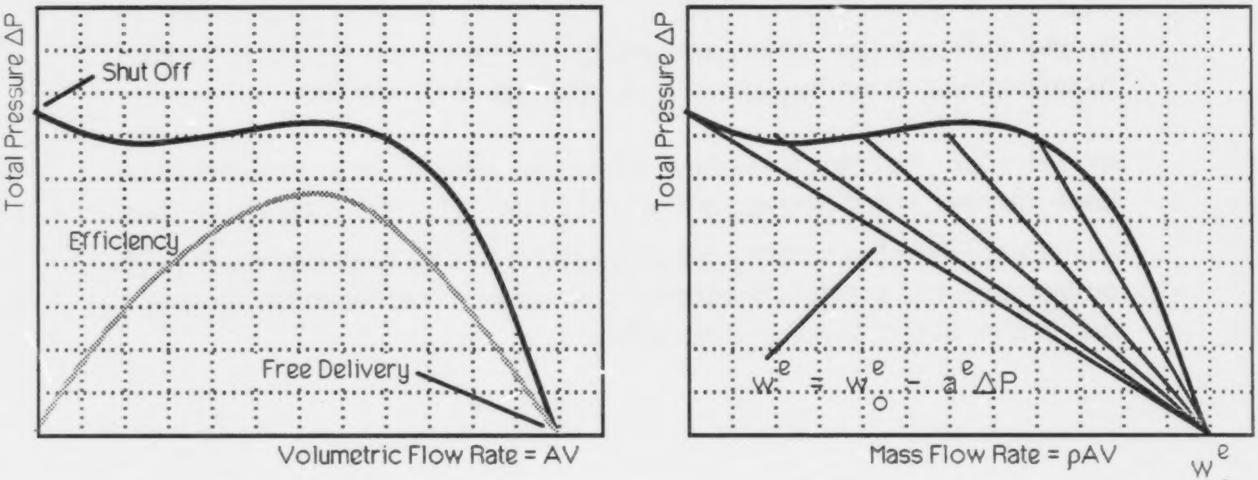
c) the flow coefficient, C^e, will also depend on the direction of flow due to the dependency of ρ on direction and the dependency of the pressure loss coefficients C_o that also, in general, depend on the direction of flow,

d) the pressure-flow coefficient matrix $[\mathbf{a}^e]$ will also be flow-direction dependent due to the flow-direction dependency of C^e and B^e,

e) equation (3.22a) is highly nonlinear due to the flow-direction dependencies, noted above, the dependency of the pressure-flow coefficient matrix $[\mathbf{a}^e]$ and the buoyancy-induced mass flow rate vector $\{\mathbf{w}_B^e\}$ on the pressure, and the dependency of density on fluid temperatures which are, in turn, dependent on the rate of flow.

3.2.2 Fan/Pump Element Equations

General operating characteristics of fans are discussed in the ASHRAE Handbook and Product Directory: 1979 Equipment [23]. Flow behavior of fans is generally described in terms of performance curves that have the following typical form;



Actual Performance Curve

Performance Curve Represented as a Family of Linear Functions

Fig. 3.5 Fan Performance Curves

Performance curves may be easily converted to pressure-mass flow curves, by scaling by the fluid density, and represented by the family of equations of the general form;

$$w^e = w_o^e - a^e \Delta P \tag{3.24}$$

where: w_o^e = the free delivery mass flow rate of the fan
a^e = $a^e(w^e)$; the fan pressure-flow coefficient
ΔP = the effective pressure drop across the fan

This family of equations has the advantage of being able to represent saddle shaped performance curves but , unfortunately, the members of the family used to "capture" the saddle shape portion of the performance curve provide very poor representations of the change of mass flow with changes of pressure and, therefore, should not be expected to perform well when used with Newton-Raphson type nonlinear solution strategies.

For nonlinear solution techniques that require the determination of the change of mass flow with changes of pressure we shall have to resort to a more restricted form of representation having;

$$a^e = a^e(\Delta P) \tag{3.25}$$

Unfortunately, a true saddle shape may not be represented with this form.

An attractive candidate for this more restricted form is offered by the following polynomial form;

$$a^e = a_1^e + a_2^e \Delta P + a_3^e \Delta P^2 + \ldots \tag{3.26}$$

or

$$w^e = w_o^e - (a_1^e \Delta P + a_2^e \Delta P^2 + a_3^e \Delta P^3 + \ldots) \tag{3.27}$$

where the coefficients, a_1^e, a_2^e, ..., would be determined by a best fit to published or measured performance curve data.

Defining fan element degrees of freedom consistent with flow resistant element degrees of freedom, as shown below, Fig. 3.6, and accounting for buoyancy effects, as in the development of the flow resistant element equations, equation (3.27) may be rewritten as;

$$w^e = w_o^e - a^e(P_m^e - P_n^e + B^e) \tag{3.28}$$

or in terms of element flow rate DOFs as;

$$\boxed{\{w_{net}^e\} = [a^e]\{P^e\} + \{w_B^e\} + \{w_o\}} \tag{3.29a}$$

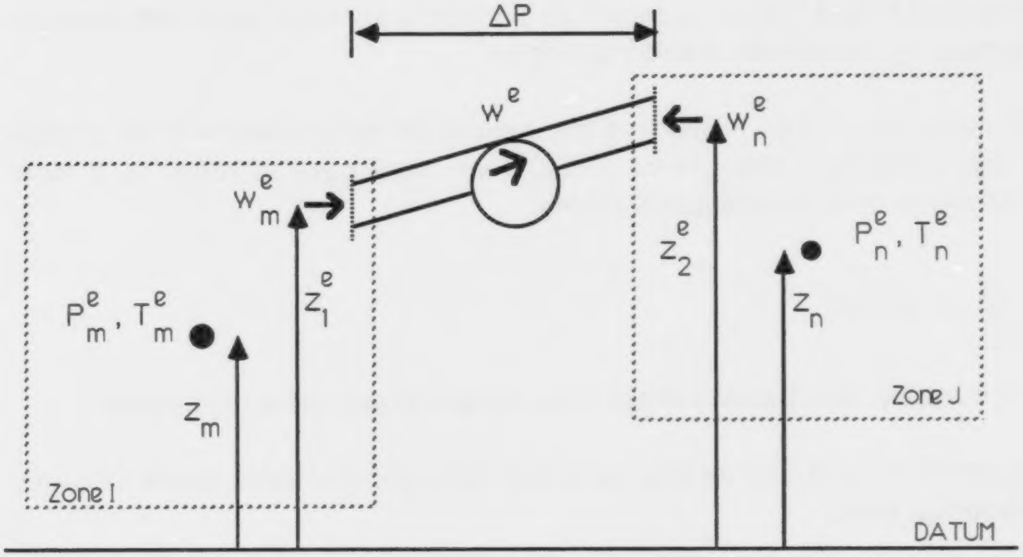


Fig. 3.6 Fan Element DOFs

where, now;

$$[a^e] = a^e \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.29b}$$

$$\{w^e_B\} = a^e B^e \{ -1 \quad 1 \}^T \tag{3.29c}$$

$$\{w_o\} = w^e_o \{ 1 \quad -1 \}^T \tag{3.29d}$$

Typically, the fan pressure-flow coefficient will be positive and therefore the matrix of fan pressure-flow coefficients, $[a^e]$, will be negative semi-definite. To account for the possibility of a system driving a fan beyond the shut off pressure - free delivery range (i.e., to account for the possibility of back flow or pressure assisted forward flow) the fan performance curve must be defined outside the conventional range of flows.

Using the polynomial form of fan performance curve, equation (3.27), we may develop analytical expressions for the variation of flow with zone pressures;

$$\frac{\partial w^e}{\partial P^e_m} = -a^e_1 - 2a^e_2(P^e_m - P^e_n + B^e) - 3a^e_3(P^e_m - P^e_n + B^e)^2 - \ldots \tag{3.30a}$$

$$\frac{\partial w^e}{\partial P^e_n} = +a^e_1 + 2a^e_2(P^e_m - P^e_n + B^e) + 3a^e_3(P^e_m - P^e_n + B^e)^2 + \ldots \tag{3.30b}$$

or, in terms of the element mass flow rate DOFs;

$$\frac{\partial\{w^e_{net}\}}{\partial\{P^e\}} = (a^e_1 + 2a^e_2(P^e_m - P^e_n + B^e) + 3a^e_3(P^e_m - P^e_n + B^e)^2 + \ldots)\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.31}$$

3.3 System Equations

Requiring conservation of mass at each flow-related node we demand;

$$\left\{ \begin{pmatrix}\text{mass generation}\\ \text{rate}\end{pmatrix} = \sum_{\substack{\text{connected}\\ \text{elements}}} \begin{pmatrix}\text{net mass flow}\\ \text{rate into element}\end{pmatrix} \right\}_{\text{system node}} \tag{3.32}$$

the element equations may be assembled to form a system of equations, that govern the flow behavior of the system, of the form;

$$\boxed{\{W\} = [A]\{P\} + \{W_B\} + \{W_o\}} \tag{3.33a}$$

where;

$$\{W\} = \{W_1, W_2, \dots W_n\}^T \tag{3.33b}$$

$$\{P\} = \{P_1, P_2, \dots P_n\}^T \tag{3.33c}$$

$$[A] = \mathop{\mathbf{A}}_{e=1}^{N_R}[a^e] + \mathop{\mathbf{A}}_{e=1}^{N_F}[a^e] \tag{3.33d}$$

$$\{W_B\} = \mathop{\mathbf{A}}_{e=1}^{N_R}\{w_B^e\} + \mathop{\mathbf{A}}_{e=1}^{N_F}\{\mathbf{w}_B^e\} \tag{3.33e}$$

$$\{W_o\} = \mathop{\mathbf{A}}_{e=1}^{N_F}\{w_o^e\} \tag{3.33f}$$

N_R, N_F = the number of flow resistance and fan elements respectively

$\mathbf{A}$ = the element assembly operator; a combination Boolean transformation and matrix summation (see section 2.2, [2] or [24] for details)

The system flow matrix, [**A**], is the sum of positive semi-definite flow resistance element matrices and negative semi-definite fan/pump element equations and, therefore, may become negative definite! Given the "1 , -1 , 1 , -1" form of the flow resistance element equations and the "-1 , 1 , -1 , 1" form of the fan/pump element equations we need only check the diagonal elements of the [**A**] matrix - if any are negative then [**A**] will be negative semi-definite otherwise it will be positive semi-definite. As will be seen in the following examples, transformation from a semi-definite to a definite matrix results upon the specification of a single nodal pressure.

3.4 Simple Examples

Two two-zone air flow examples are considered below. For these examples the density of air will be estimated using the ideal gas law as;

$$\rho = (M/R)(P/T) = \left(\frac{28.9645 \text{ kg/kgmole}}{8314.41 \text{ N-m/kgmole-°K}}\right)(P/T) = 0.00348365\,(P/T)$$

where;

ρ = density [=] kg/m^3
M = the mean molecular weight per mole of dry air
R = the universal gas constant
P = the absolute pressure [=] Pa (i.e., N/m^2)
T = the absolute temperature [=] °K

Example 1

In the first example, illustrated below, two zones are linked by two flow resistance elements, conduits in this case.

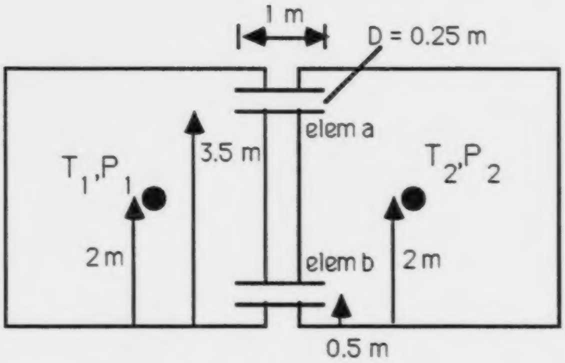


Fig. 3.7 Example 1 Flow Idealization

The temperature in zone 1 is maintained at 10 °C and that of zone 2 at 20 °C or;

$T_1 = (10 + 273.15) = 283.15$ °K

$T_2 = (20 + 273.15) = 293.15$ °K

and we seek to determine the mass flow rates through these elements and the zone pressures that will be induced by buoyancy-driven flow induced by these zone temperature differences.

Zone Densities:

- assume sea level pressure 101.325 kPa
- $\rho_{10°C} = 0.00348365 \times (101.325/(10° + 273.15°)) = 0.0012466$ kg/m^3
- $\rho_{20°C} = 0.00348365 \times (101.325/(20° + 273.15°)) = 0.0012041$ kg/m^3

Element Equations:

- Relative roughness = $\epsilon/D = 0.00015/0.25 = 0.0006$
- Friction factor: from ASHRAE Fundamentals, Chapter 2, Fig. 13; $f = 0.032$
- Cross sectional area: $A = \pi D^2/4 = \pi\ 0.25^2/4 = 0.049$ m^2
- Pressure loss coefficient: $C_o = f\ L/D = 0.032 \times 1.0 \div 0.25 = 0.128$
- Element a : connectivity 1-2
    - Assume flow is from zone 2 to zone 1 thus $\rho = \rho_{20°C}$

$$C^a = (1/2g_c\rho)(C_o/A^2{}_o) = (1/(2 \times 1 \times 0.0012041)(0.128/0.049^2) = 22137$$

$$\begin{aligned} B^a &= (g/g_c)(\rho_m(z_m - z_1{}^a) + \rho(z_1{}^a - z_2{}^a) + \rho_n(z_2{}^a - z_n)) \\ &= (9.81/1.0)(0.0012466(2-3.5) + 0.0012041(3.5-3.5) + \\ &\quad 0.0012041(3.5-2)) \\ &= -0.00062576 \end{aligned}$$

- Initial element matrices (from equations (16)): (assume $P^a{}_m = P^a{}_n$)

$$(1/C^a|P^a{}_m - P^a{}_n + B^a|)^1/_2 = (1/(22137 \times |0 + (-0.00062576)|)^1/_2 = 0.268679$$

$$\{w_B^a\} = \{\ -0.00016813 \quad 0.00016813\}^T$$

$$[a^a] = \begin{bmatrix} 0.268679 & -0.268679 \\ -0.268679 & 0.268679 \end{bmatrix}$$

- Element b: connectivity 1-2
    - Assume flow is from zone 1 to zone 2 thus $\rho = \rho_{10°C}$

$$C^b = (1/2g_c\rho)(\,C_o/A^2{}_o) = (1/(2 \times 1 \times 0.0012466))(0.128/0.049^2) = 21382$$

$$\begin{aligned} B^b &= (g/g_c)(\rho_m(z_m - z_1{}^b) + \rho(z_1{}^b - z_2{}^b) + \rho_n(z_2{}^b - z_n)) \\ &= (9.81/1.0)(0.0012466(2-0.5) + 0.0012466(0.5-0.5) \\ &\quad + 0.0012041(0.5-2)) \\ &= 0.00062576 \end{aligned}$$

- Initial element matrices (from equations (16)): (assume $P^b{}_m = P^b{}_n$)

$$(1/C^b\,|P^b{}_m - P^b{}_n + B^b|)^{1/2} = (1/(257923 \times |0 + (0.00062576)|)^{1/2} = 0.27338$$

$$\{w_B^b\} = \{\ 0.00017107\ \ -0.00017107\ \}^T$$

$$[a^b] = \begin{bmatrix} 0.27338262 & -0.27338262 \\ -0.27338262 & 0.27338262 \end{bmatrix}$$

System Equations:

The system equations may be assembled from the element equations; in this case we obtain, assuming no mass generation in the zones;

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0.54206194 & -0.54206194 \\ -0.54206194 & 0.54206194 \end{bmatrix} \begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix} + \begin{Bmatrix} 0.00000294 \\ -0.00000294 \end{Bmatrix}$$

As they stand this set of equations is singular - they describe only the pressure difference between zones. If we specify the pressure in one zone, say P_1 = 101.325, then a first estimate of P_2 may be determined; P_2 = 101.32500543. The element arrays may then be recomputed with these new estimates of P_1 & P_2 and the system equations formed and solved. By repeating this process until the results converge to acceptable accuracy a solution is obtained. For this problem we obtain, upon convergence;

P_1 = 101.3250000 Pa (i.e., as specified)
P_2 = 101.3250814 Pa

$$w^a = -0.00016922 \text{ kg/sec}$$
$$w^b = 0.00016995 \text{ kg/sec}$$

For comparison, the system equations at convergence are;

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0.54216990 & -0.54216990 \\ -0.54216990 & 0.54216990 \end{bmatrix} \begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix} + \begin{Bmatrix} 0.00000515 \\ -0.00000515 \end{Bmatrix}$$

Example 2

In this example, illustrated below, two zones are linked by a flow resistance element, identical to element "a" used in the example above, and a fan element.

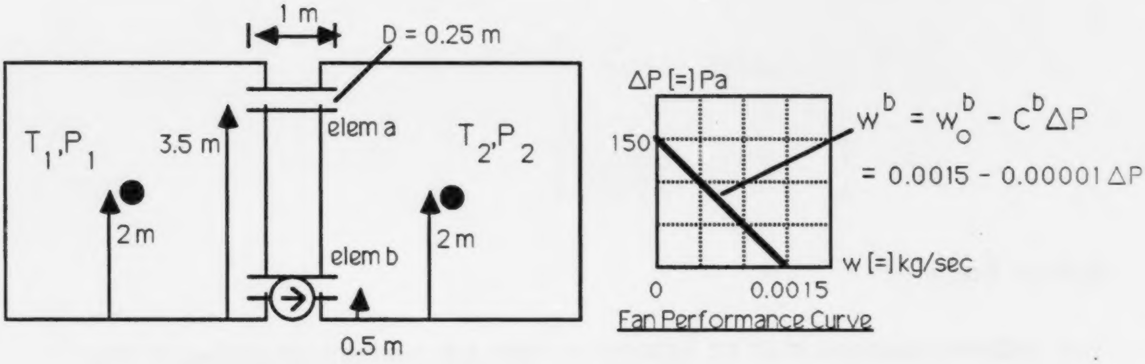

1 m
D = 0.25 m
T_1, P_1
3.5 m
2 m
elem a
elem b
T_2, P_2
2 m
0.5 m
ΔP [=] Pa
150
0
0.0015
w [=] kg/sec
$w^b = w_o^b - C^b \Delta P$
$= 0.0015 - 0.00001 \Delta P$
Fan Performance Curve

Fig. 3.8 Example 2 Flow Idealization

Again the temperature in zone 1 is maintained at 10 °C and that of zone 2 at 20 °C and we seek to determine the mass flow rates through the elements and the zone pressures that will be induced by the combined effects of buoyancy-driven and fan-driven flow.

Element Equations:

- Element a : connectivity 1-2 (as above)

  - Initial element matrices (from example 1): (assume $P^a_m = P^a_n$)

$$\{w^a_B\} = \{\ -0.00016813 \quad 0.00016813\}^T$$

$$[a^a] = \begin{bmatrix} 0.268679 & -0.268679 \\ -0.268679 & 0.268679 \end{bmatrix}$$

- Element b: connectivity 1-2

  - From the fan performance curve, above, we obtain $C^b = 0.00001$ and $w^b{}_o = 0.0015$
  - B^b is equal to that calculated for the resistance element b above: $B^b = 0.00062576$

  - Initial Element Matrices (from equations (18)): (assume $P^a{}_m = P^a{}_n$)

$$\{w_B^b\} = \{ -0.00000001 \quad 0.00000001 \}^T$$

$$\{w_o^b\} = \{ 0.0015 \quad -0.0015 \}^T$$

$$[a^b] = \begin{bmatrix} -0.00001 & 0.00001 \\ 0.00001 & -0.00001 \end{bmatrix}$$

System Equations:

The system equations may be assembled from the element equations; in this case we obtain, assuming no mass generation in the zones;

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0.26866932 & -0.26866932 \\ -0.26866932 & 0.26866932 \end{bmatrix} \begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix} + \begin{Bmatrix} -0.00016814 \\ 0.00016814 \end{Bmatrix} + \begin{Bmatrix} 0.0015 \\ -0.0015 \end{Bmatrix}$$

Again we obtain a singular set of equations that describe the pressure difference between zones. By specifying one zone pressure, say $P_1 = 101.325$, a first estimate of P_2 may be determined, in this iteration $P_2 = 101.32995727$. Again, we iteratively update element matrices with these estimates of zone pressures until results converge to acceptable accuracy. Reasonably convergent results are;

$P_1 = 101.32500000$ Pa (i.e., as specified)
$P_2 = 101.37379028$ Pa

$w^a = -0.00149407$ kg/sec
$w^b = 0.00150048$ kg/sec

For comparison, the system equations at "convergence" are;

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0.03022450 & -0.03022450 \\ -0.03022450 & 0.03022450 \end{bmatrix} + \begin{Bmatrix} -0.00001893 \\ 0.00001893 \end{Bmatrix} + \begin{Bmatrix} 0.0015 \\ -0.0015 \end{Bmatrix}$$

3.5 Solution of Flow Equations

Two classic nonlinear solution strategies and their variations;

- a) Method of Successive Substitutions or Fixed-Point Iteration
  - Direct
  - Jacobi Iteration
  - Zeid's Modified Jacobi Iteration
  - Gauss-Seidel Iteration
  - Successive Overrelaxation Method
- b) Newton-Raphson Method
  - Classic Newton-Raphson Method
  - Modified Newton-Raphson Method

and incremental formulations of these methods will be considered as candidates for solving the system of nonlinear flow equations, equations (3.33).

To set the stage for a discussion of these solution methods we rewrite the system equations, equations (3.33), in two alternate forms:

$$\{F(P)\} = [A]\{P\} + \{W_B\} + \{W_O\} - \{W\} = \{0\} \tag{3.35}$$

and

$$[A]\{P\} = \{g\} = \{W\} - \{W_O\} - \{W_B\} \tag{3.36}$$

where, it is important to be mindful that [**A**] and {$\mathbf{W}_B$} are both dependent on the state of the system pressure variables, {**P**}, and may also vary with time if the flow prblem is embedded in a dynamic thermal response problem.

3. 5.1 Successive Substitution

A class of nonlinear solution techniques have been developed and studied for equations of the form of equation (3.36) with {g} not a function of the dependent variable {P} that are based upon the use of an approximate inverse [**C**]. By adding the vector **[C]{P}** to both sides of equation (5.2);

$$[A]\{P\} + [C]\{P\} = \{g\} + [C]\{P\} \tag{3.37a}$$

or

$$\{P\} = \{P\} + [C]^{-1}\{ \{g\} - [A]\{P\} \} \tag{3.37b}$$

the governing equation is recast in a form that suggest a general iterative scheme;

$$\{P^{k+1}\} = \{P^{k}\} + [C^{k}]^{-1}\{ \{g^{k}\} - [A^{k}]\{P^{k}\} \} \tag{3.38}$$

where k is an iteration index. The term $\{ \{\mathbf{g}^k\} - [\mathbf{A}^k]\{\mathbf{P}^k\} \}$ may be thought of as a residual or error that could be monitored to evluate the convergence of the method.

The choice of the **[C]** matrix is key to the success of this approach. Clearly **[C]** must be nonsingular. Zeid shows, furthermore, that to ensure convergence **[C]** must satisfy the following condition [25],[26];

$$\| [I] - [C]^{-1}[A] \| < 1 \tag{3.39}$$

where the double bars || indicate any appropriate norm (e.g., maximum norm or Euclidean norm).

We shall consider the following alternatives, based on those developed for systems with **{g}** not a function of the dependent variable **{P}**;

Direct Iteration

The most straigtforward approach simply sets **[C]** = **[A]**;

$$\{\mathbf{P}^{k+1}\} = \{\mathbf{P}^k\} + [\mathbf{A}^k]^{-1}\{\, \{\mathbf{g}^k\} - [\mathbf{A}^k]\{\mathbf{P}^k\}\, \} \tag{3.40a}$$

or

$$\{\mathbf{P}^{k+1}\} = [\mathbf{A}^k]^{-1}\{\mathbf{g}^k\} \tag{3.40b}$$

Computationally, it is efficient to avoid inversion and instead successively solve the system of equations;

$$[A^k]\{P^{k+1}\} = \{g^k\} \tag{3.40c}$$

For systems with $\{\mathbf{g}\} \neq \{\mathbf{g}(\mathbf{P})\}$ this method often does not converge [27] and, therefore, will not be considered further.

Jacobi Iteration

Splitting the [**A**] matrix into upper and lower components as;

$$[A] = [D][\, [L] + [I] + [U]\,] \quad ; [D] = \mathrm{diag}\,(A_{ii}) \tag{3.41}$$

we set [**C**] = [**D**] to obtain;

$$\{\mathbf{P}^{k+1}\} = \{\mathbf{P}^k\} + [\mathbf{D}^k]^{-1}\{\, \{\mathbf{g}^k\} - [\mathbf{A}^k]\{\mathbf{P}^k\}\, \} \tag{3.42a}$$

or

$$\{\mathbf{P}^{k+1}\} = [\mathbf{D}^k]^{-1}\{\mathbf{g}^k\} - [\, [\mathbf{L}^k] + [\mathbf{U}^k]\,]\{\mathbf{P}^k\} \tag{3.42b}$$

For systems with $\{\mathbf{g}\} \neq \{\mathbf{g}(\mathbf{P})\}$ this method converges if $[\mathbf{A}^k]$ is strictly diagonally dominant [25],[26]. In general, [**A**] will not be strictly diagonally dominant, thus, this method is not useful here.

Zeid's Modified Jacobi Iteration

Zeid has developed a modified form of Jacobi iteration that does not require strict diagonal dominance [25],[26]. In this method we set;

$$[C^k] = \mathrm{diag}(\alpha_{ii}) \quad ; \quad \alpha_{ii} = 1/\sum_{j=1}^{n} | A^k_{ij} | \quad ; \; i=1, 2, \dots n \tag{3.43}$$

for an n x n system. The rate of convergence for this approach is linear (i.e., the error $\{\mathbf{P}^{K+1}\} - \{\mathbf{P}^{k}\}$ in each step depends linearly on the error in the last step), providing again $\{\mathbf{g}\} \neq \{\mathbf{g(P)}\}$.

Gauss-Seidel Iteration

Splitting the **[A]** matrix as before, equation (3.41), and setting **[C]** = **[D]**[**[I]** + **[L]**];

$$\{\mathbf{P}^{k+1}\} = \{\mathbf{P}^{k}\} + [\mathbf{I} + \mathbf{L}^k]^{-1}[\mathbf{D}^k]^{-1}\{\, \{\mathbf{g}^k\} - [\mathbf{A}^k]\{\mathbf{P}^k\} \,\} \tag{3.44a}$$

or

$$\{\mathbf{P}^{k+1}\} = -[\mathbf{L}^k]\{\mathbf{P}^{k+1}\} - [\mathbf{U}^k]\{\mathbf{P}^k\} + [\mathbf{D}^k]^{-1}\{\mathbf{g}^k\} \tag{3.44b}$$

For systems with $\{\mathbf{g}\} \neq \{\mathbf{g(P)}\}$ the rate of convergence of this method is linear. In indicial notation this method is;

$$r^k_i = \frac{-\sum_{j=1}^{i-1} A^k_{ij} P^{k+1}_j - \sum_{j=i}^{n} A^k_{ij} P^k_j + g^k_i}{A^k_{ii}} \tag{3.44c}$$

$$P^{k+1}_i = P^k_i + r^k_i \qquad ; \; i = 1, 2, \dots n \tag{3.44d}$$

where r is the residual that may conveniently be monitored to evaluate convergence.

Successive Overrelaxation Method

A variant of of Gauss-Seidel iteration, commonly know as the successive overrelaxation or SOR method, attempts to to accellerate convergence by scaling the residual by a *relaxation factor*, ω, as;

$$\{\mathbf{P}^{k+1}\} = \{\mathbf{P}^{k}\} + [\mathbf{I} + \mathbf{L}^{k}]^{-1}[\mathbf{D}^{k}]^{-1}\omega\{\, \{\mathbf{g}^{k}\} - [\mathbf{A}^{k}]\{\mathbf{P}^{k}\}\, \} \tag{3.45a}$$

or

$$\begin{aligned}\{\mathbf{P}^{k+1}\} = & -[\mathbf{L}^{k}]\{\mathbf{P}^{k+1}\} + (1-\omega)\{\mathbf{P}^{k}\} + [[\mathbf{L}^{k}] + \omega[\mathbf{L}^{k}]]\{\mathbf{P}^{k}\} \\ & - \omega[\mathbf{U}^{k}]\{\mathbf{P}^{k}\} + \omega[\mathbf{D}^{k}]^{-1}\{\mathbf{g}^{k}\}\end{aligned} \tag{3.45b}$$

where for ω=1.0 this reduces to Gauss-Seidel iteration. In indicial notation this method is;

$$r_i^k = \frac{-\sum_{j=1}^{i-1} A_{ij}^k P_j^{k+1} - \sum_{j=i}^{n} A_{ij}^k P_j^k + g_i^k}{A_{ii}^k} \tag{3.45c}$$

$$P_i^{k+1} = P_i^k + \omega r_i^k \qquad i = 1, 2, \dots n \tag{3.45d}$$

This method can only converge for $0 < \omega < 2$ [28].

For the governing flow equations, equations (3.36), the forcing vector {**g**} will, in general, depend upon the dependent variable {**P**} and thus the convergence rates and conditions on convergence noted above can, at best, provide only guidelines; we are not in a position at this time to say much about the convergence of these adaptations of classical fixed-point methods.

Upon closer examination, however, we note that $\{\mathbf{g}\} = \{\mathbf{W}_B(\mathbf{P})\} + \{\mathbf{W}_O\}$, the sum of a bouyancy-related flow vector, that is pressure dependentand a fan-related flow vector that is not. If the flow is largely forced (i.e., by fans or wind-induced pressure), so that the bouyancy-related flow is relatively small, then we should expect these adapted methods to behave as theory predicts.

3.5.2 Newton-Raphson Iteration

The following development of the Newton-Raphson Method and its variants is based largely on the formulation presented by Bjork and Anderson [28].

Using Taylor's formula, generalized for a system of n equations, we may approximate the function {**F**(**P**)}, from equation (3.35), from its value at a nearby vector {$\mathbf{P}^k$} as;

$$\{F(P)\} = \{F(P^k)\} + [F'(P^k)]\{\ \{P\} - \{P^k\}\ \} + O(\,||\{P\}-\{P^k\}||^2) \qquad (3.46)$$

where **F'** is the Jacobian defined as;

$$[F'(P^k)] = \frac{\partial\{F(P)\}}{\partial\{P\}}\bigg|_{\{P\}=\{P^k\}} \qquad (3.47a)$$

or

$$F'_{ij}(P^k) = \frac{\partial F_i(P)}{\partial P_j}\bigg|_{\{P\}=\{P^k\}} \qquad (3.47b)$$

Equation (3.46) leads naturally to the general form of the popular Newton-Raphson iterative method;

$$[F'(P^k)]\{\Delta P^{k+1}\} = -\ \{F(P^k)\} \qquad (3.48a)$$

$$\{P^{k+1}\} = \{P^k\} + \{\Delta P^{k+1}\} \qquad (3.48b)$$

where, again, k is the iteration index. Given an initial guess {$\mathbf{P}^o$} sufficiently close to the solution the method will converge at a quadratic rate.

The high rate of convergence has made this approach popular, but the method involves the formation of the n x n entries of the Jacobian and the solution of an n x n system of equations at each iteration - tasks that become computationally prohibitive as n increases.

<ins>Evaluation of the Jacobian</ins>

For the problem at hand, equation (3.35), the Jacobian involves the evaluation of;

$$\frac{\partial\{F(P)\}}{\partial\{P\}}\Big|_{\{P\}=\{P^k\}} = \frac{\partial(\,[A]\{P\} + \{W_B\}\,)}{\partial\{P\}}\Big|_{\{P\}=\{P^k\}} \tag{3.49a}$$

or

$$\frac{\partial\{F(P)\}}{\partial\{P\}}\Big|_{\{P\}=\{P^k\}} = \mathop{\mathbf{A}}_{e=1}^{N_R} \frac{\partial\{w^e_{net}\}}{\partial\{P^e\}}\Big|_{\{P^e(P^k)\}} + \mathop{\mathbf{A}}_{e=1}^{N_F} \frac{\partial\{w^e_{net}\}}{\partial\{P^e\}}\Big|_{\{P^e(P^k)\}} \tag{3.49b}$$

that is, the Jacobian is simply evaluated as an element assembly sum of the element Jacobians evaluated at the element pressures {$\mathbf{P}^e$} corresponding to the current iterative estimate of the system pressure {$\mathbf{P}^k$}.

Modified Newton-Raphson Iteration

To avoid some of the computational expense of forming and solving the Newton-Raphson equations, (3.48), at each iteration, one may reform [**A**] , [**F'**], and {$\mathbf{W}_B$} only occaissonally, say every m steps, as;

$$[F'(P^p)]\{\Delta P^{k+1}\} = -[A^p]\{P^k\} - \{W^p_B\} - \{W_o\} + \{W\} \qquad ;\, k=p, \ldots\, p+1 \tag{3.50}$$

This modified Newton-Raphson method saves computation but at a cost of convergence. Attempts have been made to compensate for a lower convergence rate by using an *overrelaxation factor* , ω, applied to the residual, ΔP, calculated at each step, as;

$$\{P^{k+1}\} = \{P^k\} + \omega\{\Delta P^{k+1}\} \tag{3.51}$$

the choice of ω is likely to be "problem dependent and the experience of the analyst will be crucial" [29]; values of ω ≈ 2.0 are often used.

3.5.3 Incremental Formulation

When flow is driven primarily by fans (i.e., when the buoyancy-related flow is relatively small) it may prove useful to approach a solution incrementally by considering incremental increases in fan free delivery flow $\{\mathbf{W}_o\}$. After Zienkkiewicz [27] we rewrite equation (3.35) in the form;

$$[A]\{P\} + \{W_B\} + \lambda\{\{W_o\} - \{W\}\} = \{0\} \tag{3.52}$$

and solve a series of nonlinear problems, incrementally increasing λ to 1.0; the solution at each increment may then be used as the initial guess of the solution at the next increment. For increments of λ suitably small we may be assured that the initial guesses of the incremental solutions will be sufficiently close to the solution to guarantee convergence, _if_ the solution to;

$$[A]\{P\} + \{W_B\} = \{0\} \tag{3.53}$$

is available (e.g., if $\{\mathbf{W}_B\}$ is a zero vector) or can be computed.

For m increments of λ;

$$^{m}\lambda = 1/m\ ,\ 2/m\ ,\ \ldots\ m/m \tag{3.54}$$

the Gauss-Seidel method, with overrelaxation becomes, in indicial notation;

$$^{m}r_i^{\,n+1} = \frac{-\sum_{j=1}^{i-1} {}^{m}A_{ij}^{k}\, {}^{m}P_j^{k+1} - \sum_{j=i}^{n} {}^{m}A_{ij}^{k}\, {}^{m}P_j^{k} - {}^{m}W_{B\,i}^{k} + {}^{m}\lambda(W_i - W_{o\,i})}{{}^{m}A_{ij}^{k}} \tag{3.55a}$$

$$^{m}P_i^{k+1} = {}^{m}P_i^{k} + \omega\, {}^{m}r_i^{k} \qquad ;\ i=1,\ 2,\ \ldots\ n\ , \tag{3.55b}$$

and the modified Newton-Raphson method, also with overrelaxation, becomes;

$$[F'({}^{m}P^{p})]\{\Delta {}^{m}P^{k+1}\} = -[{}^{m}A^{p}]\{{}^{m}P^{k}\} - \{{}^{m}W_{B}^{k}\} - {}^{m}\lambda\{\,\{W_{o}\} - \{W\}\,\} \qquad (3.56a)$$

$$\{{}^{m}P^{k+1}\} = \{{}^{m}P^{k}\} + \omega\{\Delta {}^{m}P^{k+1}\} \qquad ; k = p, p+1, \ldots p+l \qquad (3.56b)$$

with updating of system arrays every l+1 steps. In both cases, at each increment, m, one iterates on k.

4. Summary and Directions of Future Work

Summary

The theoretical basis of a building indoor air quality model has been presented that provides for;

a) contaminant dispersal analysis of nonreactive contaminants, and

b) mechanical, wind, and thermally-driven air flow analysis

in multi-zone buildings of arbitrary complexity. It has been shown that both contaminant dispersal analysis and air flow analysis equations may be assembled from element equations that govern the behavior of discrete flow elements in the building airflow system. The general, qualitative character of these equations has been discussed and efficient numerical methods have been presented for their solution.

This theoretical work extends the work of others (e.g., [18], [30],[31]) in that;

a) for both contaminant dispersal and flow analysis;

- the governing equations are assembled from element equations so that systems of arbitrary complexity may be considered, existing computational strategies based upon element assembly methods may be employed, and formal analysis of the system equations is possible from the new perspective of the element assembly operation,
- efficient numerical methods have been identified for the practical solution of the governing equations, and

b) for contaminant dispersal analysis;

- filtering of contaminants has been accounted for,
- practical methods of accounting for unsteady flow conditions have been identified,

- the qualitative analysis of the multi-zone contaminant dispersal equations has been extended demonstrating, importantly, that the conservation of total air flow, alone, in a building idealization (without the need to place special qualifications on zones isolated from exterior air infiltration, e.g., [31] p. 225) leads to nonsingular M-matrices that may be efficiently factored to LU form, and

c) for flow analysis;

- element equations governing passive resistance air flow paths has been extended to allow consideration of a variety of simple and complex air flow paths,

- element equations governing fan-driven air flow have been developed that may readily be assembled, with the general resistance element, to allow analysis of building air flow systems of arbitrary complexity, and

- low-flow conditions have been modeled consistently with existing flow theory in such a way that should help to avoid convergence problems experienced by others (some preliminary computational studies indicate success here).

In PART II of this report a program, CONTAM86, is presented that implements the contaminant dispersal portion of the theory and examples of its application, that provide preliminary validation, are discussed.

Directions of Future Work

In the near future, work will be directed toward the two general areas considered thus far - contaminant dispersal analysis and air flow analysis. In addition, the inverse contaminant dispersal problem will be considered (i.e., the determination of airflows, in a multi-zone building system, from knowledge of zonal concentrations due to known excitations). In the distant future, hopefully, the coupled multi-zone building flow and thermal analysis problem and its integration with the contaminant dispersal analysis problem will be considered by integrating the building thermal analysis methods developed earlier [2] with the methods introduced here.

In the area of contaminant dispersal analysis the present theory will be extended;

a) through the development of *reaction elements* , to allow modeling of the dispersal of single and multiple reactive contaminants, and

b) through the development of *one-dimensional convection-diffusion flow elements* , to allow modeling of the details of contaminant dispersal for flow in duct-type flow passages.

In addition, an attempt will be made to develop elements to model the dynamics of contaminant adsorption and absorption into the building fabric and furnishings.

The flow analysis theory will be implemented to provide computational tools that may be used in an integrated manner with the contaminant dispersal analysis tools presently available in CONTAM86. An attempt will be made to evaluate the several nonlinear solution strategies, discussed in section 3.5, so that guidelines for their use may be formulated.

The inverse problem of determining multi-zone air flow rates from measured contaminant concentration and generation rate data (e.g., as used in tracer gas flow measuring techniques) will, also, be addressed. That the inverse problem is inherently an *ill-conditioned* problem (i.e., small errors in concentration and generation rate data typically result in large errors in estimated airflow quantities) is not well appreciated, therefore, this effort will place an emphasis on determination of the conditioning of the inverse problem, for specific applications, and identification of strategies of formulating the inverse problem to minimize ill-conditioning. Coupling the formulation and solution of specific inverse analysis problems with the determination of their conditioning provides, as an additional benefit, a means to place error bounds on the estimates of airflows. Again, the inverse problem will be formulated using an element assembly approach, to allow consideration of systems of arbitrary complexity, and implemented so as to augment the computational tools available and presently under development for dispersal and flow analysis.

PART I References

[1] McNall, P., Walton, G., Silberstein, S., Axley, J., Ishiguro, K., Grot, R., & Kusuda, T., Indoor Air Quality Modeling Phase I Report: Framework for Development of General Models, NBSIR 85-3265, U. S. Dept. of Commerce, National Bureau of Standards, October 1986

[2] Axley, J.W., DTAM1: A Discrete Thermal Analysis Method for Building Energy Simulation: Part I Linear Thermal Systems with DTAM1 Users' Manual, (presently under review for publication by the National Bureau of Standards, Building Environment Division, Center for Building Technology)

[3] Bird, R.B., Stewart, W.E., & Lightfoot, E.N., Transport Phenomena, John Wiley & Sons, Inc.,N.Y.,1960

[4] Zienkiewicz, O.C. & Morgan, K., Finite Elements and Approximation, John Wiley & Sons, N.Y., 1983, pp. 154-157

[5] Funderlic, R.E. & Plemmons, R.J., "LU Decomposition of M-Matrices by Elimination Without Pivoting", Linear Algebra and Its Applications, Vol 41, pp. 99-110, Elsevier, North Holland,1981

[6] Plemmons, R.J., Nonegative Matrices in the Mathematical Sciences, Chapter 6: M-Matrices, Academic Press, 1979, p.137 and 147

[7] Ibid. p 156

[8] Funderlic, R.E., Neumann, M., & Plemmons, R.J., "LU Decomposition of Generalized Diagonally Dominant Matrices", Numer. Math., Springer-Verlag, Vol 40, p 57-69, (1982) Theorem 4

[9] Graybill, F.A., Matrices with Applications in Statistics, Wadsworth, Belmont CA., 1983 Section 11.4

[10] Noble, B., Applied Linear Algebra, Prentice-Hall, N.J., 1969

[11] Strang, G., Linear Algebra and Its Application, Academic Press, N.Y., 1980

[12] Ibid. p 213

[13] Wilkinson, J.H. & Reinsch, C., Handbook for Automatic Computation: Volume II: Linear Algebra, Springer-Verlag, 1971 - Part II: The Algebraic Eigenvalue Problem

[14] ibid., Contribution II/12: "Solution to the Eigenproblem by a Norm Reducing Jacobi Type Method", by P.J. Eberlein & J. Boothroyd

[15] Taylor, Robert L., "HEAT", A Finite Element Computer Program for Heat-Conduction Analysis, Report 75-1, Prepared for: Civil Engineering Laboratory, Naval Construction Battalion Center, Port Hueneme, California, May 1975

[16] Huebner, K.H., & Thornton, E., The Finite Element Method for Engineers: Second Edition, John Wiley & Sons, New York, 1982

[17] Hughes, T.J., "Analysis of Some Fully-Discrete Algorithms for the One-Dimensional Heat Equation", International Journal for Numerical Methods in Engineering, Vol. 21, John Wiley & Sons, 1985

[18] Walton, G., Thermal Analysis Research Program Reference Manual, NBSIR 83-2655, U.S. Dept. of Com., National Bureau of Standards, Gaithersburg, MD, March 83

[19] Carnahan, B., Luther, H.A., Wilkes, J.O., Applied Numerical Methods, John Wiley & Sons, 1969

[20] ASHRAE, ASHRAE Handbook: 1985 Fundamentals, Chapter 33 Duct Design,, ASHRAE, Atlanta, GA, 1985

[21] ASHRAE, ASHRAE Handbook: 1985 Fundamentals, Chapter 14 Airflow Around Buildings, ASHRAE, Atlanta, GA, 1985

[22] ASHRAE, ASHRAE Handbook: 1985 Fundamentals, Chapter 2 Fluid Flow, ASHRAE, Atlanta, GA, 1985

[23] ASHRAE, ASHRAE Handbook & Product Directory: 1979 Equipment, Chapter 3 Fans, ASHRAE, New York, 1979

[24] Bathe, K.J., Finite Element Procedures in Engineering Analysis, Second Edition, John Wiley & Sons, New York, 1982

[25] Zeid, I., "Fixed-Point Iteration to Nonlinear Finite Element Analysis. Part I: Mathematical Theory and Background," International Journal for Numerical Methods in Engineering, Vol.21, No.11, John Wiley & Sons, New York, 1985

[26] Zeid, I., "Fixed-Point Iteration to Nonlinear Finite Element Analysis. Part II: Formulation and Implementation," International Journal for Numerical Methods in Engineering, Vol.21, No.11, John Wiley & Sons, New York, 1985

[27] Zienkiewicz, O.C., The Finite Element Method, 3rd Edition, McGraw-Hill, London, 1977, p. 452

[28] Bjork, A., Anderson, N., Numerical Methods, Prentice-Hall, Inc., New Jersey, 1974

[29] Chow, Y.K. & Kay, S., "On the Aitken Acceleration Method for Nonlinear Problems", Computers & Structures, Vol. 19, No. 5/6, 1984, pp. 757-761

[30] Sinden, F.W., "Multi-Chamber Theory of Air Infiltration", Building and Environment, Vol. 13, Pergamon Press, Great Britain, 1978, pp. 21-28

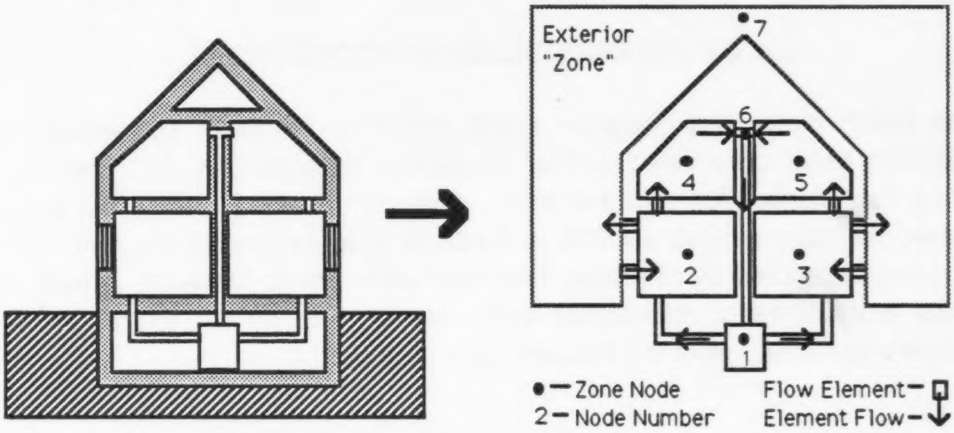
[31] Sandberg, M, "The Multichamber Theory Reconsidered from the Viewpoint of Air Quality Studies", Building and Environment, Vol. 19, No. 4, Pergamon Press, Great Britain, 1984, pp. 221-233

5. General Instructions

The program CONTAM86 is a command processor[5-1]; it responds to commands in the order that they are presented and processes data associated with each command. Commands may be presented to the program interactively, using keyboard and monitor, or through the use of command/data input files; that is to say, it offers two modes of operation - interactive and batch modes.

For most practical problems of contaminant dispersal analysis the batch mode of operation will be preferred. For these problems, analysis involves three basic steps;

Step 1: Idealization of the Building System and Excitation



Actual Building

Air-Flow System Idealization

Fig. 5.1 Idealization of the Building System and Excitation

Idealization of the building flow system involves

a) discretization of the system as an assemblage of appropriate flow elements connected at system nodes,
b) identification of boundary conditions, and
c) numbering of system nodes optimally (i.e., to minimize the bandwidth -

---

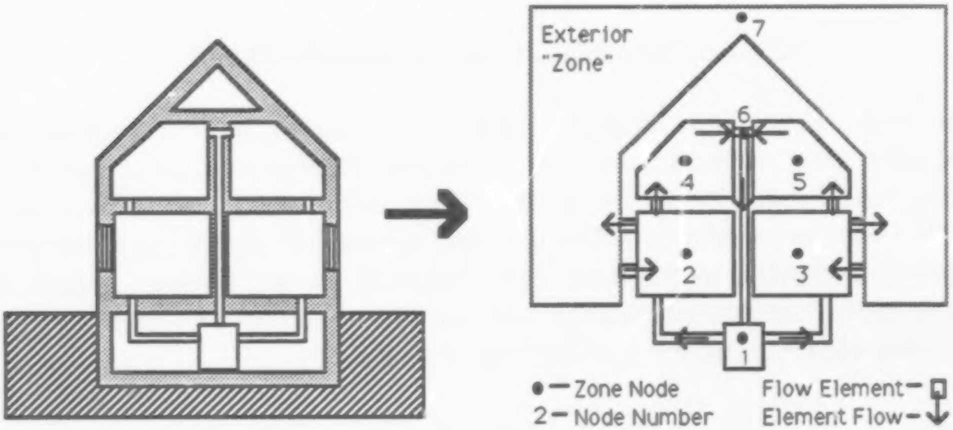
[5-1] CONTAM86 is written in FORTRAN 77. The complete source code for the program may be found in the attached appendix.

5. General Instructions

The program CONTAM86 is a command processor[5-1]; it responds to commands in the order that they are presented and processes data associated with each command. Commands may be presented to the program interactively, using keyboard and monitor, or through the use of command/data input files; that is to say, it offers two modes of operation - interactive and batch modes.

For most practical problems of contaminant dispersal analysis the batch mode of operation will be preferred. For these problems, analysis involves three basic steps;

Step 1: Idealization of the Building System and Excitation



Actual Building

Air-Flow System Idealization

Fig. 5.1 Idealization of the Building System and Excitation

Idealization of the building flow system involves

a) discretization of the system as an assemblage of appropriate flow elements connected at system nodes,
b) identification of boundary conditions, and
c) numbering of system nodes optimally (i.e., to minimize the bandwidth -

---

[5-1] CONTAM86 is written in FORTRAN 77. The complete source code for the program may be found in the attached appendix.

node number difference - of system equations).

The excitation (i.e., specified contaminant concentrations and generation rates) may be modeled to be steady or defined in terms of arbitrary time histories. For the latter case initial conditions of nodal contaminant concentration will have to also be specified.

Step 2: Preparation of Command/Data Input File

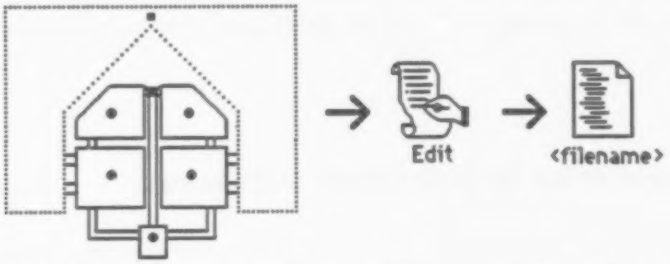


Fig. 5.2 Preparation of Input Command/Data File

In the batch mode, the program reads ASCII text files of commands and associated data, collected together in distinct data groups, that define the building flow idealization and excitation. The command/data input file may be prepared with any available ASCII text editing program and given a file name, <filename>, specified by the user. The <filename> must, however, consist of 8 or less alphanumeric characters and can not include an extension (i.e., characters separated from the filename by a period, ".").

Step 3: Execution of CONTAM86

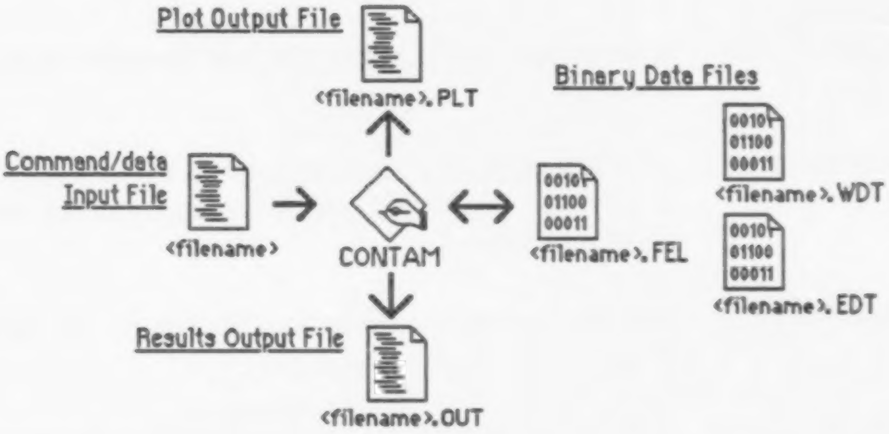


Fig. 5.3 Execution of CONTAM86

CONTAM86 is then executed. Initially CONTAM86 will be in the interactive mode. To enter the batch mode the command "SUBMIT F=<filename>" may be used to "submit" the command/data input file to the program. The program will then proceed to form element and system arrays and compute the solution to the posed problem. CONTAM86 reads the ASCII command/data input file and creates an ASCII (i.e., printable) output file <filename>.OUT. The results of an analysis, <filename>.OUT, may be conveniently reviewed using an ASCII editor and, from the editor, portions or all of the results may be printed out. Key response results are also written to the ASCII file <filename>.PLT in a format that may easily be transferred to some spreadsheet and plotting programs (i.e., data values within each line are separated by the tab character) for plotting or subsequent processing.

File Summary

Depending upon the commands processed, CONTAM86 will also create a variety of binary files for out-of-core storage needed for subsequent processing. A summary of files read and created includes;

Files Read

<filename>	an ASCII input file specified by the user that contains

	commands and associated data

Files Created

<filename>.OUT	a printable ASCII output file that contains analysis results
<filename>.PLT	an ASCII output file that contains key analysis results in a form that may be transferred to spreadsheet and/or plotting programs
<filename>.FEL	a binary file used for out-of-core storage of flow element data
<filename>.WDT	a binary file used for out-of-core storage of element flow time history data
<filename>.EDT	a binary file used for out-of-core storage of excitation time history data

In the interactive mode <filename> is set to the default value of "CONTAM86" and commands are read from the keyboard. A help command, "HELP" or "H", will produce a screen listing of all available commands.

6. Command Conventions

Commands and their associated data (if any) may be single-line or multiple-line command/data groups.

Single-Line Commands

Single line command/data groups begin with the command keyword and may have any number of associated data items identified by data identifies of the typical form;

```
COMMAND  A=n1,n2,n3  B=n4  C=n5,n6  D=c1c2c3
```

where n1,n2,n3,... is numeric data and c1c2c3 is character data. In this example the keyword **COMMAND** is the command keyword and the data identifiers are **A=**, **B=**, **C=**, and **D=**.

Multiple-Line Commands

Multiple-line command/data groups are delimited by the command keyword and the keyword **END** and may have any number of data subgroups terminated by the symbol "<" within. They have the typical form of;

```
COMMAND  A=n1,n2
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
n1  I=n2,n3,n4  B=n5  C=c1c2c3c4
<
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
n1,n2,n3  D=n4,n5,n6  E=n7  F=c1c2c3
<
c1c2c3c4c5c6
END
```

Classes of Commands

Two general groups of commands are available, the "Intrinsic Commands" and the "CONTAM86 Commands". The "Intrinsic Commands" are useful, primarily, in the interactive mode allowing the user to examine system arrays generated by the "CONTAM86 commands" and save them for further processing by the CAL-80 command processor or other command processors based on the CALSAP in-core management routines [1]. The "CONTAM86 Commands" provide contaminant dispersal analysis operations.

Command/data Lines

Normally the line length (i.e., the number of character and spaces on a line) is limited to 80. A backslash "\" at the end of information on any line will, however, allow the next line to be interpreted as a continuation of the first line providing an effective line length of 160.

Use of the symbol "<" within in any line indicates the end of information on that line. Information entered to the right of this symbol is ignored by the program and may, therefore, be used to annotate a command/data input file.

An asterisk "*" at the beginning of any line will cause the line to be echoed as a comment on the console and to the output file. Lines marked in this way may, then, be used to annotate the output file and help indicate the progress of computation when using the batch mode of operation.

Data Identifiers

Data identifiers and their associated data may be placed in any order within each line of the command/data group with the exception that the first line of a command/data group must begin with the command keyword. In some instances data may not be associated with a data identifier, such data must be placed first in a line.

Data

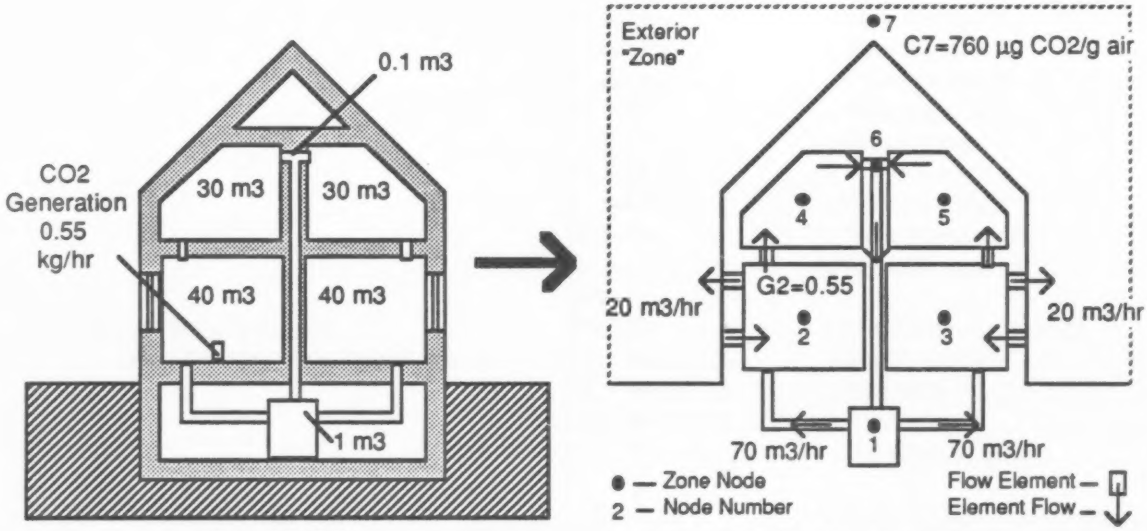
Decimal points are not required for real numeric data. Scientific notation of the form nnE+nn or nn.nnE+nn (e.g., 5.79E-13) may be used. Simple arithmetic expressions employing the conventional operators +, –, *, and / may be used. The order of evaluation is sequential from left to right - unlike FORTRAN or other programing languages where other "precedence" rules are used.

If fewer data values are supplied than required the missing data will assumed to be zero, blank, or set to default values as appropriate.

7. Introductory Example

For purposes of contaminant dispersal analysis the specific command/data groups that need to be included in a command/data input file will depend upon the details of the flow system idealization, the nature of the excitation, and the type of analysis to be computed. A specific introductory example, should however, provide some useful insight into the more general aspects of contaminant dispersal analysis using CONTAM86

Consider the two-story residence with basement shown, in section, below. In this residence interior air is circulated by a forced-air furnace and exterior air infiltrates the house through leaks around the two first floor windows. The flow system may be idealized using flow elements to model the ductwork, room-to-room, and infiltration flow paths as shown below.


0.1 m3
CO2 Generation 0.55 kg/hr
30 m3
30 m3
40 m3
40 m3
1 m3
Exterior "Zone"
7
C7=760 µg CO2/g air
6
4
5
G2=0.55
2
3
20 m3/hr
20 m3/hr
1
70 m3/hr
70 m3/hr
● — Zone Node
2 — Node Number
Flow Element —
Element Flow —

Actual Building

Air-Flow System Idealization

Fig 7.1 Hypothetical Residential Example

For this building idealization we shall consider the hypothetical problem of determining the steady state distribution of CO_2 generated by a kerosene heater placed in room "2", distributed by the furnace flow system operated at constant conditions, and diluted by infiltration at a constant rate. The CO_2

generation rate is assumed to be 0.55 kg/hr, exterior CO_2 concentration is assumed to be 760 µg CO_2/ g air, and the assumed air volumetric flow rates are indicated on the drawings above.

The CONTAM86 command/data file to complete this steady state analysis is listed below. Command/data groups needed to complete a time constant analysis and dynamic analysis for this building idealization are presented as examples in the reference section of this manual.

Command/data File for Residential Example

Note: CONTAM86 keywords and identifies are displayed in boldface below.

Description	Command/data File	
Column	1	
Comments:	*	
Comments	* Six-Zone (7-Node) Example	
Comments	* Units: kg, m, hr	
Comments	* Concentration [=] kg-CO2/kg-air	
Comments	* Generation rate [=] kg-CO2/hr	
Comments	*	
System Definition:	**FLOWSYS N**=7	< System has 7 Nodes
Boundary Conditions	7 **BC**=C	< Ext. "Zone" Conc. Spec.
	END	
Flow Element Data:	**FLOWELEM**	
Element Number & Connectivity,	1 **I**=1,2	< Flow Element 1
	2 **I**=1,3	< Flow Element 2
	3 **I**=7,2	< Flow Element 3
	4 **I**=2,7	< Flow Element 4
	5 **I**=7,3	< Flow Element 5
	6 **I**=3,7	< Flow Element 6
	7 **I**=2,4	< Flow Element 7
	8 **I**=3,5	< Flow Element 8
	9 **I**=4,6	< Flow Element 9
	10 **I**=5,6	< Flow Element 10
	11 **I**=6,1	< Flow Element 11
	END	
Steady State Solution:	**STEADY**	< (Air Density 1.2 kg/m3)
Flow Element Mass Flow Rates	1,2 **W**=70*1.2	< Supply Ducts
	3,6 **W**=20*1.2	< Infiltration
	7,10 **W**=70*1.2	< Return Loop
	11 **W**=140*1.2	< Main Return Duct
	<	
Contaminant Excitation	2 **CG**=0.55	< Node 2: Generation Rate
	7 **CG**=0.000760	< Node 7: Ext. CO2 Conc.
	END	
Return to Interactive Mode	**RETURN**	

Details are given on the following pages for each of CONTAM86's command/data groups.

8. Command Reference

8.1 Intrinsic Commands

8.1.1 HELP

The command **HELP**, or simply **H**, will produce a list of all available commands, in abbreviated form.

8.1.2 ECHO

The command **ECHO-ON** acts to cause computed results normally directed to the results output file to be echoed to the screen. The command **ECHO-OFF** turns this feature off. At start-up CONTAM86 is set to **ECHO-ON**. Selective use of **ECHO-ON** and **ECHO-OFF** can speed computation as writing results to the screen consumes a significant amount of time.

8.1.3 LIST

The command **LIST**, or simply **L**, will produce a list of all arrays currently in the in-core array database.

8.1.4 PRINT A=<arrayname>

The command **PRINT A=**<arrayname> or simply **P A=**<arrayname> will "print" array named <arrayname>, a one-to four character name, to the screen.

8.1.5 DIAGRAM A=<arrayname>

The command **DIAGRAM A=**<arrayname> will "print" a diagram of array named <arrayname>, a one-to four character name, to the screen indicating position of zero and nonzero terms. (Character arrays can not be diagramed.)

8.1.6 SUBMIT F=<filename>

The command **SUBMIT F=**<filename> will cause the program to switch to batch mode and read all subsequent commands from the file <filename>.

8.1.7 RETURN

The command **RETURN** returns the operation of the program from batch mode to interactive mode. **RETURN** or **QUIT** will normally be the last line of batch command/data input files.

8.1.8 QUIT

The command **QUIT** or simply **Q** terminates execution of the program and returns the user to the control of the operating system.

8.2 CONTAM86 Commands

The following conventions will be used for the command definitions presented in this section;

- an ellipses, '. . . ', indicates unlimited repetition of similar data items or data lines within a data subgroup
- square brackets, [...], indicate optional data,
- numeric data is indicated by lower case n, as n1,n2, ... , and
- character data by lower case c, as c1.

8.2.1 FLOWSYS

The size of the flow system and boundary conditions of system nodes are defined with the following command/data group;

```
FLOWSYS  N=n1
n2,n3,n4  BC=c1
. . .
END
```

where; n1 = the number of flow nodes
n2,n3,n4 = first node, last node, node increment of a series of nodes with identical boundary conditions
c1 = boundary condition code; **C** for concentration prescribed nodes; **G** for generation prescribed nodes; (default = **C**)

The direct species mass generation rate or the species concentration - but not both - may be specified at each node to establish boundary conditions of prescribed contaminant generation or concentration.

If this boundary condition data is omitted all nodes will be assumed to be species mass generation rate DOFs. Typically, nodes associated with outdoor environmental conditions will be assigned specific contaminant concentrations and nodes associated with indoor air zones will be assigned specific species generation rates although zero generation rates will often be appropriate for these nodes.

See the introductory example presented earlier for an example of the use of this command.

8.2.2 FLOWELEM

Two-node flow elements may be added to the flow system assemblage with the following command/data group;

```
FLOWELEM
n1  I=n2,n3  GEN=n4  E=n5
...
END
```

where; n1 = the element number
n2, n3 = the element node numbers
n4 = generation increment (default = 1)
n5 = the element filter efficiency (default = 1.0)

Element data must be supplied in numerical order. Omitted data is automatically generated by incrementing the preceding node numbers by the current generation increment. Generated elements will have the properties of the current element.

See the introductory example presented earlier for an example of the use of this command.

8.2.3 STEADY

The response of the system to steady contaminant generation with steady element mass flow may be computed with the following command/data group;

```
STEADY
n1,n2,n3  W=n4
```

```
. . .
<
n5,n6,n7  CG=n8
. . .
END
```

where; n1,n2,n3 = first element, last element, element number increment of a series of elements with identical mass flow rates
n4 = element total mass flow rate; (default = 0.0)
n5,n6,n7 = first node, last node, node increment of a series of nodes with identical excitation
n8 = contaminant concentration or contaminant generation rate, as appropriate to the boundary condition of the node; (default = 0.0)

Net total mass flow rate at each system node will be reported, but computation will not be aborted if net mass flow is nonzero. The analyst must assume the responsibility to check continuity of mass flow from these reported values.

See the introductory example presented earlier for an example of the use of this command.

8.2.4 TIMECONS

System time constants, nominal and actual, may be computed with the following command/data group;

```
TIMECONS [E=n1]
n2,n3,n4  W=n5
. . .
<
n6,n7,n8  V=n9
. . .
END
```

where; n1 = optional convergence parameter, epsilon ; (default = machine precision)
n2,n3,n4 = first element, last element, element number increment of a

series of elements with identical mass flow rates
n5 = element total mass flow rate; (default = 0.0)
n6,n7,n8 = first node, last node, node increment of a series of nodes with identical volumetric masses
n9 = nodal volumetric mass; (default = 0.0)

The *nominal* time constants are computed for each node as the quotient of the nodal volumetric mass divided by the total air flow out of a zone. The *actual* time constants are computed using an eigenanalysis routine that is a variant of Jacobi iteration adapted for nonsymmetric matrices [2]. It should be noted that the actual time constants are likely to be very different from the nominal time constants for systems having well-coupled zones. Be advised: eigenanalysis of the flow system matrices is a time consuming task.

Example

To determine the time constants associated with the building idealization presented earlier, in the introductory example, the following command/data group would have to be added to the command/data file.

```
TIMECONS          < (Air Density 1.2 kg/m3)
1,2   W=70*1.2          < Supply Ducts
3,6   W=20*1.2          < Infiltration
7,10  W=70*1.2          < Return Loop
11    W=140*1.2         < Main Return Duct
<
1     V=1.2*1.0         < Node 1      Vol. Mass
2,3   V=1.2*40.0        < Nodes 2 & 3 Vol. Mass
4,5   V=1.2*30.0        < Nodes 4 & 5 Vol. Mass
6     V=1.2*0.1         < Node 6      Vol. Mass
7     V=1.2*1.0E+06     < Node 7  Ext. Vol. Mass
END
```

8.2.5 Dynamic Analysis

The response of the system, including transients, to general dynamic excitation, may be computed using the command **DYNAMIC**. The dynamic solution procedure used is driven by discrete time histories of excitation and element mass flow data that must first be generated with the commands **FLOWDAT** and **EXCITDAT**. (In future releases of CONTAM element mass flow data may also be generated by a detailed flow analysis of the flow system.)

8.2.5.1 FLOWDAT

Discrete time histories of element mass flow rate may be defined, in step-wise manner, from given element mass flow data, as illustrated below;

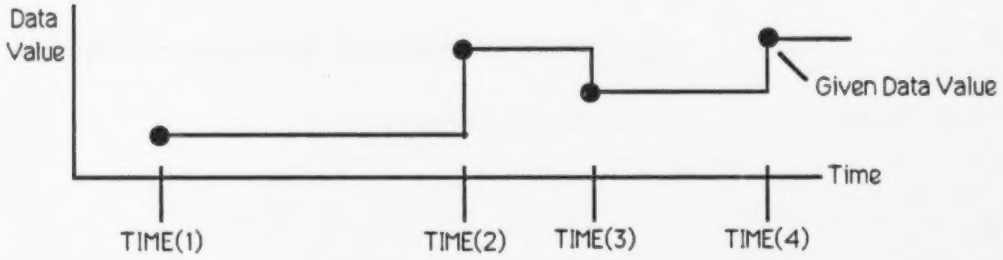


Fig. 8.1 Arbitrarily Defined Time History Data

or, alternatively, discrete time histories of element mass flow data, defined in a step-wise manner at equal time-step intervals along piece-wise linear segments, may be generated from given element mass flow data over a time range defined by an initial time, T_i, a final time, T_f, and a generation time increment, ΔT, as illustrated below;

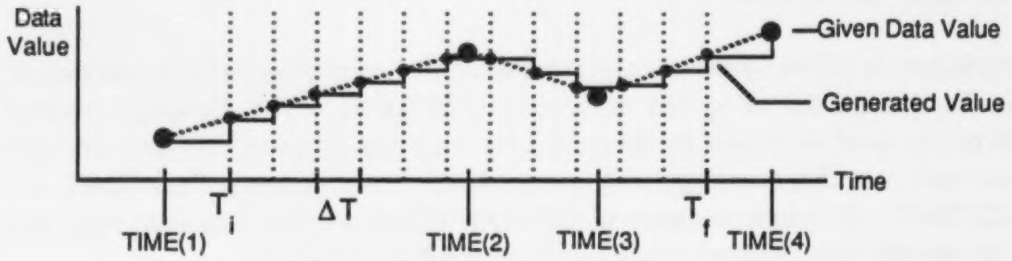


Fig. 8.2 Equal-Time-Step-Generated Time History Data

using the following command/data group;

```
FLOWDAT  [T=n1,n2,n3]
TIME=n4
n5,n6,n7  W=n8
. . .
<
TIME=n4                     [additional TIME data, as necessary, to define the
complete
n5,n6,n7  W=n8              excitation time history]
. . .
<
END
```

where; n1,n2,n3 = initial time, final time, time step increment used for the piece-wise linear generation option
n4 = time value for subsequent data subgroups
n5,n6,n7 = first element, last element, element number increment of a series of elements with identical mass flow data
n8 = prescribed element mass flow: (default = 0.0)

If data values n1,n2,n3 are specified, step-wise time histories will be generated from the given data, along piece-wise linear segments as illustrated in Fig. 8.2 above, otherwise the given data will be used directly, as illustrated in Fig. 8.1 above.

At least two "TIME" data subgroups must be provided. **FLOWDAT** writes the generated time history to the file <filename>.WDT so that this data may subsequently be accessed by the command **DYNAMIC**.

8.2.5.2 EXCITDAT

Discrete time histories of excitation data may be defined in the two ways discussed above for the **FLOWDAT** command using the following command/data group;

```
EXCITDAT  [T=n1,n2,n3]
TIME=n4
n5,n6,n7  CG=n8
. . .
<
TIME=n4                          [additional TIME data, as necessary, to define the
n5,n6,n7  CG=n8                   complete excitation time history]
. . .
<
END
```

where; n1,n2,n3 = initial time, final time, time step increment used for the piece-wise linear generation option
n4 = time value for subsequent data subgroups
n5,n6,n7 = first node, last node, node number increment of a series of nodes with identical excitation data
n8, = prescribed contaminant concentration <u>or</u> prescribed contaminant generation rate (as appropriate to node boundary condition): (default = 0.0)

If data values n1,n2,n3 are specified, step-wise time histories will be generated, from the given data, along piece-wise linear segments as illustrated in Fig. 8.2 above, otherwise the given data will be used directly, as illustrated in Fig. 8.1 above.

At least two "TIME" data subgroups must be provided. **EXCITDAT** writes the generated time history to the file <filename>.EDT so that it may subsequently be accessed by the command **DYNAMIC.**

8.2.5.3 DYNAMIC

The response of the system to excitation defined by the **EXCITDAT** command, using the prescribed element flow data defined by the **FLOWDAT** command, may be computed using the following command/data group;

```
DYNAMIC
T=n1,n2,n3  [THETA=n4]  [PI=n5]  [PS=n6]
n7,n8,n9  V=n10
. . .
<
n7,n8,n9  IC=n11
. . .
END
```

where; n1,n2,n3 = initial time, final time, time step increment
n4 = integration parameter, θ, where $0 \leq \theta \leq 1$; (default = 0.75) instability may result for $\theta < 0.5$,
n5 = response results print interval; (default = 1)
n6 = plot file results scale factor; if not equal to 0.0, an ASCII file, <filename>.PLT, of concentration response results will be created with values scaled by the factor n6
n7,n8,n9 = first node, last node, node increment of a series of nodes with identical data
n10 = nodal volumetric mass; (default = 0.0)
n11 = initial nodal concentration; (default = 0.0)

The response is computed using the predictor-corrector method discussed in PART I of this report. With this method, the system flow matrix is updated at the discrete times used to define element flow rate time histories and the system excitation is updated at the discrete times used to define excitation time histories, as illustrated below;

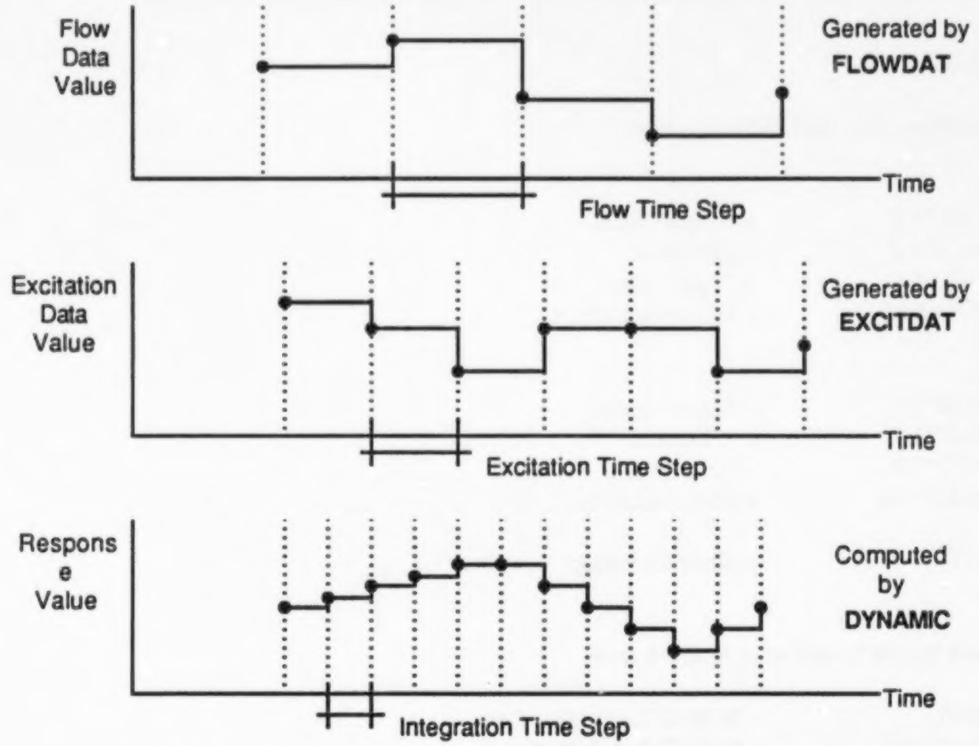

Flow Data Value
Generated by FLOWDAT
Time
Flow Time Step
Excitation Data Value
Generated by EXCITDAT
Time
Excitation Time Step
Respons e Value
Computed by DYNAMIC
Time
Integration Time Step

Fig. 8.3 Flow and Excitation Driven Dynamic Solution Procedure

The accuracy of the computed response is, therefore, dependent upon the choice of the flow data time step, the excitation data time step, and the integration time step chosen by the analyst. Furthermore, the flow data and excitation data time steps may be nonconstant. The analyst should, therefore, consider investigating the effects of the choice of these time step variables to gain a sense of the error they induce.

8.2.5.4 Dynamic Analysis Example

To provide an example of a command/data sequence needed for dynamic analysis we may consider an extension to the introductory example presented earlier; the analysis of the dynamic response of the given building system, under conditions of constant air flows, to a step change in CO_2 generation. Specifically, to consider the case where the kerosene heater is turned on and then turned off 133 minutes later the following command/data group would have to be added to the command/data file used in the introductory example.

```
FLOWDAT
*
* Element flow rates modeled as constant.
*
TIME=0
1,2    W=70*1.2          < Supply Ducts
3,6    W=20*1.2          < Infiltration
7,10   W=70*1.2          < Return Loop
11     W=140*1.2         < Main Return Duct
<
TIME=5
1,2    W=70*1.2          < Supply Ducts
3,6    W=20*1.2          < Infiltration
7,10   W=70*1.2          < Return Loop
11     W=140*1.2         < Main Return Duct
END
EXCITDAT                 < Nodal Excitation
TIME=0
*
* Kerosene heater turned on at time = 0 mins.
*
2  CG=0.55               < Node 2: Generation Rate
7  CG=0.000760           < Node 7: Ext. CO2 Conc.
<
TIME=133/60
*
* Kerosene heater turned off at time = 133 mins.
*
2  CG=0.0                < Node 2: Generation Rate
7  CG=0.000760           < Node 7: Ext. CO2 Conc.
<
TIME=5
2  CG=0.0                < Node 2: Generation Rate
7  CG=0.000760           < Node 3: Ext. CO2 Conc.
END
DYNAMIC
T=0,4,0.1  PS=1.0E+6     < Time-step; Plot Scale
1    V=1.2*1.0           < Node 1      Vol. Mass
2,3  V=1.2*40.0          < Nodes 2 & 3  Vol. Mass
4,5  V=1.2*30.0          < Nodes 4 & 5  Vol. Mass
6    V=1.2*0.1           < Node 6      Vol. Mass
7    V=1.2*1.0E+06       < Node 7  Ext. Vol. Mass
<
1,7  IC=0.000760         < Initial Concentrations
END
```

8.2.6 RESET

The command **RESET** resets the system in preparation for a new analysis problem (i.e., key internal variables are re-initialized, contaminant dispersal analysis system arrays are deleted from memory, and existing binary files are deleted from disk storage). The system is automatically reset, if necessary, upon execution of the FLOWSYS command.

RESET may be used to delete binary files that would otherwise be left on disk at the termination of the program.

9. Example Problems

9.1 Single Zone Examples

It is useful to first consider a single zone building air flow system that exchanges indoor air with the exterior environment. Such a single zone system may be modeled as an assemblage of two flow elements, corresponding to inlet and exhaust flow paths, connected to two system nodes, corresponding to the inside air zone and the exterior environment "zone" as illustrated below;

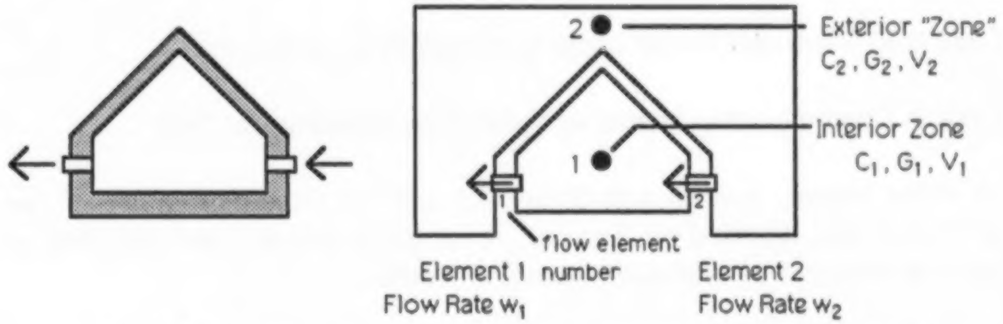


Fig. 6.1 A Single Zone Building and Corresponding Flow Model

The equations governing this simplest flow system have the following general form;

$$\begin{bmatrix} w_1 & -w_2 \\ -w_1 & w_2 \end{bmatrix} \begin{Bmatrix} C_1 \\ C_2 \end{Bmatrix} + \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix} \begin{Bmatrix} \frac{dC_1}{dt} \\ \frac{dC_2}{dt} \end{Bmatrix} = \begin{Bmatrix} G_1 \\ G_2 \end{Bmatrix} \qquad (9.1)$$

where;

w_1 , w_2 = intake and exhaust element flow rates, respectively
C_1 , C_2 = interior and exterior contaminant concentrations, respectively
V_1, V_2 = interior and exterior volumetric masses, respectively
G_1, G_2 = interior and exterior contaminant generation rates, respectively.

From a consideration of mass continuity we require $w_1 = w_2 = w$ and therefore equations (9.1) may be rewritten in expanded form as;

$$w\,C_1 - w\,C_2 + V_1\frac{dC_1}{dt} = G_1 \qquad (9.2a)$$

$$-w\,C_1 + w\,C_2 + V_2\frac{dC_2}{dt} = G_2 \qquad (9.2b)$$

With these equations in hand we shall proceed to consider three cases;

Case 1: Contaminant Decay under Steady Flow Conditions

Case 2: Contaminant Decay under Unsteady Flow Conditions

Case 3: Contaminant Dispersal Analysis of an Experimental Test

In all three cases, system characteristics will be based on those of an experimental test reported by Traynor, et. al [3] involving measurements of pollutant emissions from portable kerosene heaters.

9.1.1 Case 1: Contaminant Decay under Steady Flow Conditions

Consider the particularly simple, and familiar, case of contaminant decay from some initial value, $C_1(t=0)$, under steady flow conditions, w = constant, with concentration in the exterior environment maintained at the zero level, $C_2 = 0$. Under these conditions equation (9.2a) simplifies to;

$$w\,C_1 + V_1\frac{dC_1}{dt} = 0 \qquad (9.3)$$

whose exact solution is;

$$C_1 = C_1(t=0)\, e^{-\frac{t}{(V_1/w)}} \qquad (9.4)$$

(the quotient (V_1/w) is commonly know as the time constant of the system).

This exact solution is compared, below, to approximate solutions generated with the program CONTAM using integration time steps of $\Delta t = 2.0$, 1.0, and 0.5 hrs with $C_1(t=0) = 1.0 \times 10^{-6}$ kg / kg air, $V_1 = 31.87$ kg, and $w = 12.75$ kg/hr (i.e., 0.4 air changes per hour).

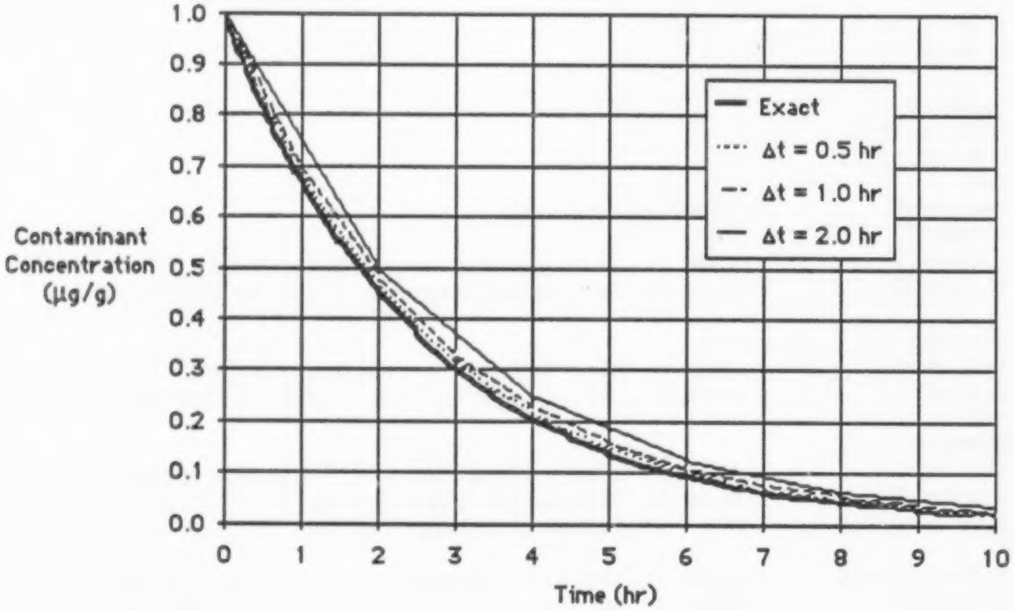


Fig. 9.2 Single Zone Model: Contaminant Decay under Steady Flow Conditions

The accuracy of the general predictor-corrector method used to approximate the response of this system is related to the time constant of the system being studied. In this case the time constant is (31.87 kg/12.75 kg/hr) = 2.5 hr. From the results of this single study, then, it appears that using an integration time increment equal to a fraction of the system time constant will assure practically accurate results.

Case 1: Command/data Input File for $\Delta t = 0.5$

The CONTAM command/data file and resulting results output file are listed below. It should be noted that a large number was used for the volumetric mass of the exterior "zone" to affect a model of a practically infinite contaminant sink.

```
FLOWSYS N=2     < Single-Zone (2-Node) Example
2 BC=C
END
FLOWELEM
1 I=1,2         < Flow Element 1
2 I=2,1         < Flow Element 2
END
FLOWDAT         < Element Mass Flow Rates [=] kg/hr
TIME=0
1 W=12.75
2 W=12.75
<
TIME=15
1 W=12.75
2 W=12.75
END
EXCITDAT        < Nodal Excitation
TIME=0
1 CG=0.0        < Node 1: Zero Generation Rate [=] kg/hr
2 CG=0.0        < Node 2: Zero Concentration   [=] kg CO2/kg
<
TIME=15
1 CG=0.0        < Node 1: Zero Generation Rate [=] kg/hr
2 CG=0.0        < Node 2: Zero Concentration   [=] kg CO2/kg
END
DYNAMIC
T=0,10,0.5      < Initial Time, Final Time, Time Step Increment
1 V=31.87       < Node 1: Volumetric Mass [=] kg
2 V=1.0E+9      < Node 2: Volumetric Mass [=] kg
<
1 IC=1.0E-06    < Node 1: Initial Concentration [=] kg CO2/kg
2 IC=0.0        < Node 2: Initial Concentration [=] kg CO2/kg
END
RETURN
```

Case 1: Results Output File

```
-----------------------------------------------------------------------------
| CONTAM: Contaminant Dispersal Analysis for Building Systems               |
-----------------------------------------------------------------Ver-10-86---
                                                     Jim Axley - Cornell & NBS
                                                                MTOT:   50000

 ==== FLOWSYS: FLOW SYSTEM CONTROL VARIABLES

      Number of flow system nodes ......    2

   == Node Boundary Conditions

      Negative Eqtn-# = concentration-prescribed boundary.
      Positive Eqtn-# = generation-prescribed boundary.
```

```
     Node    Eqtn-#  Node    Eqtn-#  Node    Eqtn-#  Node    Eqtn-#  Node    Eqtn-#
       1       1       2      -2

==== FLOWELEM: FLOW ELEMENTS

      Elem     I-Node    J-Node    Filter Efficency
         1          1         2      .000
         2          2         1      .000

==== FLOWDAT: ELEMENT FLOW TIME HISTORY DATA

  == Generation Control Variables

      Initial time ..................    .000
      Final time ....................    15.0
      Time step increment ...........    15.0

  == Element Mass Flow Time History Data

  == Time:    .000

      Elem    Value    Elem    Value    Elem    Value    Elem    Value    Elem    Value
         1    12.7        2    12.7

  == Time:    15.0

      Elem    Value    Elem    Value    Elem    Value    Elem    Value    Elem    Value
         1    12.7        2    12.7

==== EXCITDAT: EXCITATION TIME HISTORY DATA

  == Generation Control Variables

      Initial time ..................    .000
      Final time ....................    15.0
      Time step increment ...........    15.0

  == Nodal Excitation Time History Data

  == Time:    .000

               "*" = independent DOFs       "U" = undefined DOFs.

     Node    Value    Node    Value    Node    Value    Node    Value     Node    Value
        1    .000       2*    .000

  == Time:    15.0

               "*" = independent DOFs       "U" = undefined DOFs.

     Node    Value    Node    Value    Node    Value    Node    Value     Node    Value
        1    .000       2*    .000

==== DYNAMIC: DYNAMIC SOLUTION
```

```
== Solution Control Variables

   Initial time ..................     .000
   Final time ....................     10.0
   Time step increment ...........     .500
   Integration parameter: alpha ...    .750
   Results print interval .........       1

== Nodal Volumetric Mass

          "*" = independent DOFs       "U" = undefined DOFs.

  Node   Value    Node   Value    Node   Value    Node   Value     Node   Value
    1    31.9       2*   0.100E+10

== Initial Conditions: Nodal Concentrations

          "*" = independent DOFs       "U" = undefined DOFs.

  Node   Value    Node   Value    Node   Value    Node   Value     Node   Value
    1    0.100E-05  2*    .000

== Element Flow Rate Update ================================== Time:   .000

   Elem    Value   Elem    Value   Elem    Value   Elem    Value   Elem    Value
     1     12.7      2     12.7

== Net Total Mass Flow

          "*" = independent DOFs       "U" = undefined DOFs.

  Node   Value    Node   Value    Node   Value    Node   Value     Node   Value
    1    .000       2*    .000

== Excitation Update ========================================= Time:   .000

          "*" = independent DOFs       "U" = undefined DOFs.

  Node   Value    Node   Value    Node   Value    Node   Value     Node   Value
    1    .000       2*    .000

== Time Step Estimate for Initial Conditions

-- NOTE: Estimated time step to limit error to approx. 5.00% is:  .925
         Specified time step is:  .500

== Response ================================================== Time:   .500

          "*" = independent DOFs       "U" = undefined DOFs.

  Node   Value    Node   Value    Node   Value    Node   Value     Node   Value
    1    0.826E-06  2* -0.593E-30

== Response ================================================== Time:   1.00

          "*" = independent DOFs       "U" = undefined DOFs.
   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.682E-06  2* -0.187E-29

 == Response ================================================== Time:    1.50

          "*" = independent DOFs        "U" = undefined DOFs.

   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.564E-06  2* -0.372E-29

 == Response ================================================== Time:    2.00

          "*" = independent DOFs        "U" = undefined DOFs.

   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.466E-06  2* -0.603E-29

 == Response ================================================== Time:    2.50

          "*" = independent DOFs        "U" = undefined DOFs.

   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.385E-06  2* -0.874E-29

 == Response ================================================== Time:    3.00

          "*" = independent DOFs        "U" = undefined DOFs.

   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.318E-06  2* -0.118E-28

                            ---- ( et cetera ) ----

 == Response ================================================== Time:    10.0

          "*" = independent DOFs        "U" = undefined DOFs.

   Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
     1   0.219E-07  2* -0.686E-28
```

9.1.2 Case 2: Contaminant Decay under Unsteady Flow Conditions

To investigate the consequence of unsteady flow on the nature of the behavior of the "real" system and the numerical characteristics of its simulation we shall extend Case 1 by considering the decay of a contaminant under conditions of linearly increasing flow rates, that is to say with;

$$w = w^0 t \qquad ; t \geq 0.0 \tag{9.5}$$

The decay problem is now governed by the equation;

$$w^0 t C_1 + V_1 \frac{dC_1}{dt} = 0 \qquad C_1(t=0) = 1.0 \qquad (9.6a)$$

. or

$$w^0 t\, dt = V_1 \frac{dC_1}{C_1} \qquad C_1(t=0) = 1.0 \qquad (9.6b)$$

The second form, with variables t and C_1 separated, may be integrated directly to obtain the exact solution;

$$C_1 = 1.0\, e^{-\frac{t^2}{(2V_1/w^0)}} \qquad (6.7)$$

Again this exact solution is compared to approximate solutions generated with the program CONTAM86, below. For this case, however, the numerical consequences of both integration time step, Δt, and step-wise approximation of the unsteady flow, Δtw, (i.e., the flow approximation time step) can be considered. (The solution was generated for $V_1 = 31.87$ kg, and $w^0 = 3.187$ kg/hr^2.)

In this case, using an integration time step equal to the flow approximation time step, $\Delta t = \Delta tw$, (i.e., updating the system flow matrix at each time step) provides practically accurate results for even the relatively large time step of 2.0 hr (see Figure 9.3). Updating the system flow matrix every other time step introduces an offset error equal to the flow approximation time step (when compared to results obtained with updating at each time step) for the first time step that is gradually diminished with each successive time step (see Figure 9.4). This initial offset error results because of the initial zero flow condition; in other cases the initial error would not be expected to be as great.

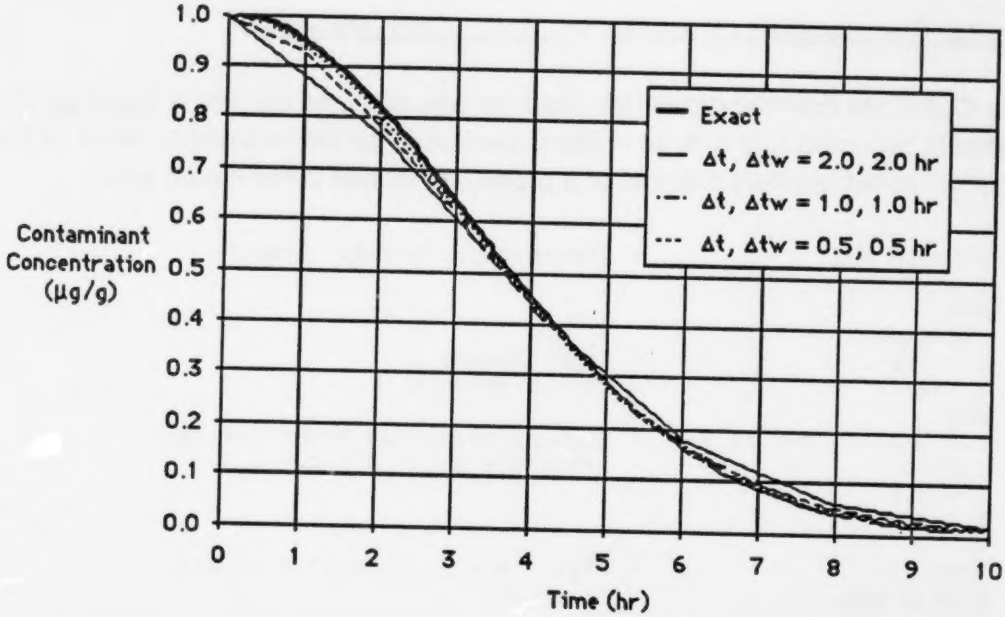


Fig. 9.3 Single Zone Contaminant Decay under Unsteady Flow Conditions with Flow Updating at Each Integration Time Step

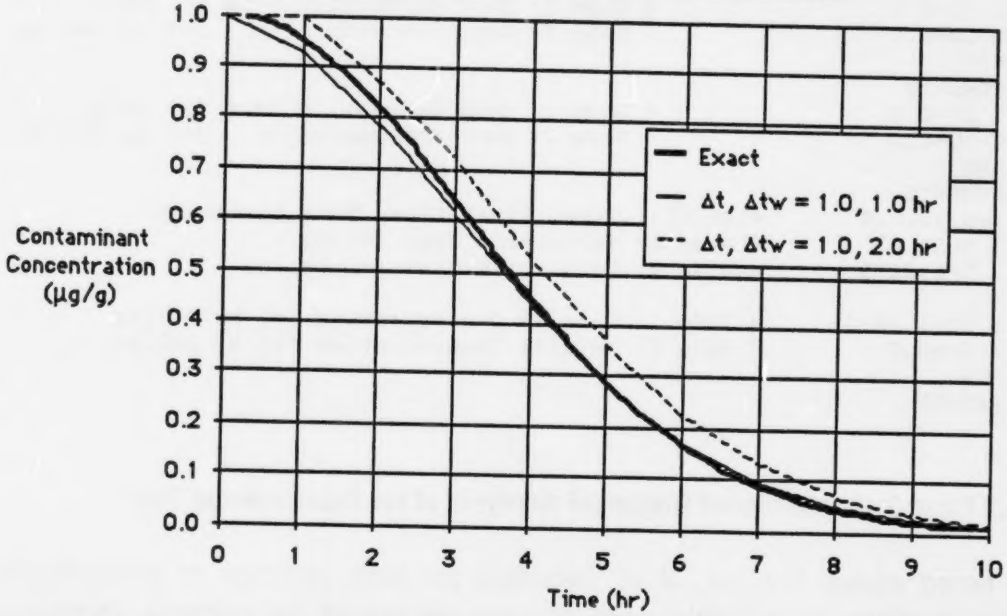


Fig. 9.4 Single Zone: Contaminant Decay under Unsteady Flow Conditions with Flow Updating at Every Other Integration Time Step

Case 2: Command/data Input File for $\Delta t = 1.0$ and $\Delta tw = 2.0$

The CONTAM command/data file used for one of these studies is listed below. It should be noted that a large number was used for the volumetric mass of the exterior "zone" to affect a model of a practically infinite contaminant sink.

```
FLOWSYS N=2             < Single-Zone (2-Node) Example
2 BC=C
END
FLOWELEM
1 I=1,2                 < Flow Element 1
2 I=2,1                 < Flow Element 2
END
FLOWDAT     T=0,12,2    < Element Mass Flow Rates [=] kg/hr
TIME=0                  < t=0 : w = 3.187 X 0.0 = 0.0
1 W=0.0
2 W=0.0
<
TIME=12                 < t=12 : w = 3.187 X 12 = 38.244
1 W=38.244
2 W=38.244
END
EXCITDAT                < Nodal Excitation
TIME=0
1 CG=0.0                < Node 1: Zero Generation Rate [=] kg/hr
2 CG=0.0                < Node 2: Zero Concentration   [=] kg CO2/kg
<
TIME=15
1 CG=0.0                < Node 1: Zero Generation Rate [=] kg/hr
2 CG=0.0                < Node 2: Zero Concentration   [=] kg CO2/kg
END
DYNAMIC
T=0,10,1.0       < Initial Time, Final Time, Time Increment
1 V=31.87        < Node 1: Volumetric Mass [=] kg
2 V=1.0E+9       < Node 2: Volumetric Mass [=] kg
<
1 IC=1.0E-06     < Node 1: Initial Concentration [=] kg CO2/kg
2 IC=0.0         < Node 2: Initial Concentration [=] kg CO2/kg
END
RETURN
```

9.1.3 Case 3: Contaminant Dispersal Analysis of an Experimental Test

As noted above Traynor, et.al. reported the time variation of contaminant concentrations in a single zone system generated by portable kerosene heaters. In this example the variation of NO concentration, C_1 , in a single zone system is computed, using measured properties of the system and NO generation rate, and compared to experimental results. The properties of the system and excitation used in the model are as follows;

V_1: single zone volumetric mass = 31.87 kg (based on the reported volume of 27 m^3 and an assumed air density of 1.18 03 kg/m^3 corresponding to 26 °C and 1 atm)

G_1 : NO generation rate = 0.000186 kg/hr constant for one hour, zero thereafter (based on the product of the reported emission rate of 23.7 µg/kJ times the fuel consumption of 7830 kJ/hr)

V_2 : exterior "zone" volumetric mass = 1.0 – 10^9 kg (infinite sink modeled as a large number)

C_2 : exterior "zone" ambient concentration = 0.0 kg NO/kg air (based on reported initial conditions)

w: air mass flow rate = 12.43 kg/hr (based on reported air change rate of 0.39 ACH)

Experimental results are compared below, Figure 9.66, to analytical results using two integration time steps. The reported generation rate time history is shown in Figure 9.5.

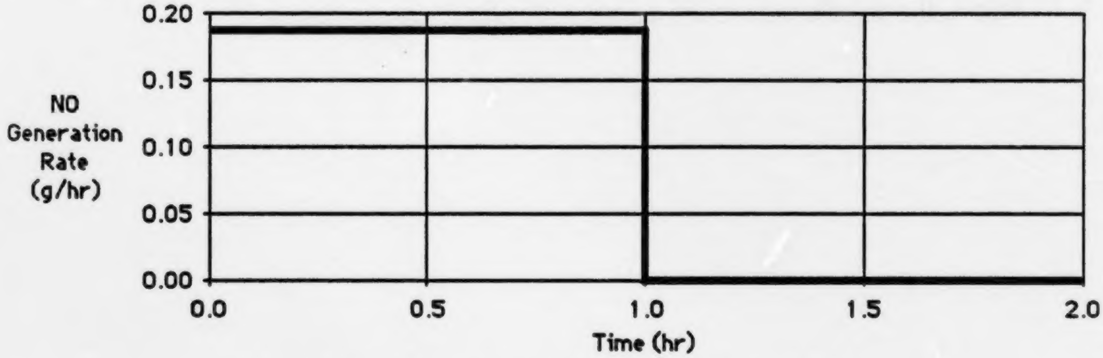


Fig. 9.5 NO Generation Rate Time History Models

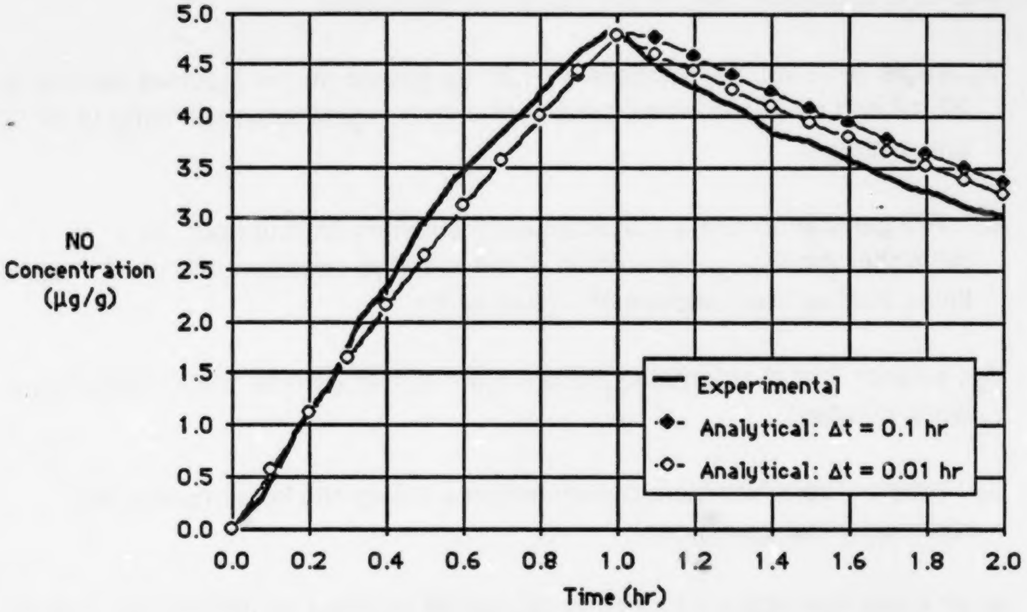


Fig. 9.6 Single Zone: NO Contaminant Dispersal Analysis of an Experimental Test

Traynor, et. al. also studied the time variation of CO_2 concentration generated by portable kerosene heaters in the same single zone system. Experimental results for one of these studies are compared to analytical results below, Figure 9.7. Again, the predicted results agree well with measured data.

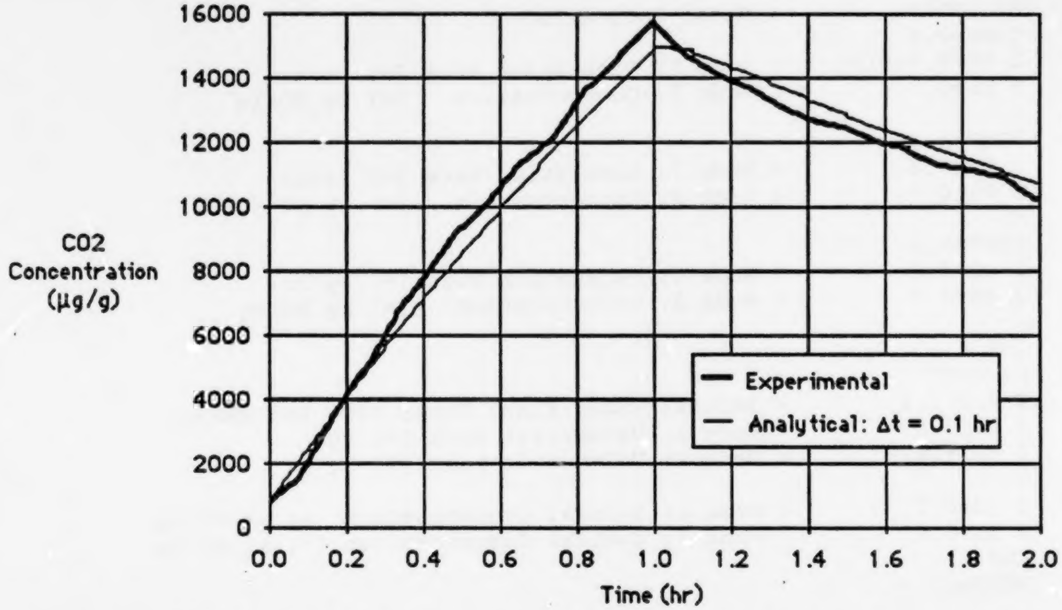


Fig. 9.7 Single Zone: CO_2 Contaminant Dispersal Analysis of an Experimental Test

Case 3: Command/data Input File for Δt = 0.10, NO Generation Rate History #1

The CONTAM command/data file used for one of these studies is listed below. It should be noted that a large number was used for the volumetric mass of the exterior "zone" to affect a model of a practically infinite contaminant sink.

```
FLOWSYS N=2       < Single-Zone (2-Node) Example
2 BC=C
END
FLOWELEM
1 I=1,2           < Flow Element 1
2 I=2,1           < Flow Element 2
END
FLOWDAT           < Element Mass Flow Rates [=] kg/hr
TIME=0
1 W=12.43         < 0.39 Air Changes Per Hour
2 W=12.43
<
TIME=3.5
1 W=12.43         < 0.39 Air Changes Per Hour
2 W=12.43
END
```

```
EXCITDAT          < Nodal Excitation
TIME=0.0
1 CG=0.000186     < Node 1: Generation Rate [=] kg/hr
2 CG=0.0          < Node 2: Concentration   [=] kg NO/kg
<
TIME=1.0
1 CG=0.0          < Node 1: Generation Rate [=] kg/hr
2 CG=0.0          < Node 2: Concentration   [=] kg NO/kg
<
TIME=3.5
1 CG=0.0          < Node 1: Generation Rate [=] kg/hr
2 CG=0.0          < Node 2: Concentration   [=] kg NO/kg
<
END
DYNAMIC
T=0,2,0.1         < Initial Time, Final Time, Time Increment
1 V=31.87         < Node 1: Volumetric Mass [=] kg
2 V=1.0E+9        < Node 2: Volumetric Mass [=] kg
<
1 IC=0.0          < Node 1: Initial Concentration [=] kg NO/kg
2 IC=0.0          < Node 2: Initial Concentration [=] kg NO/kg
END
RETURN
```

9.2 Two Zone Example

In another study Traynor et. al. [4] studied the variation of contaminant concentration generated by portable kerosene heaters in a multi-room residence that was modeled as a two-zone flow system. In this study a kerosene heater was placed in a master bedroom that was allowed to exchange air with the rest of the house and the exterior environment under a variety of test conditions. Here we shall attempt to model one of these tests that allowed relatively large flow rates between the master bedroom and the rest of the house.

For this test Traynor et. al. report the time history of the flow rate between the master bedroom and the rest of the house, the whole-house infiltration rate, and the volumes of the master bedroom and the rest of the house. The contaminant generation rate produced by the kerosene heater was reported in the earlier study discussed above. The heater was operated for a period of 133 minutes. Based on these reports a two-zone building and its corresponding flow model may be formulated as illustrated below (Figure 9.8).

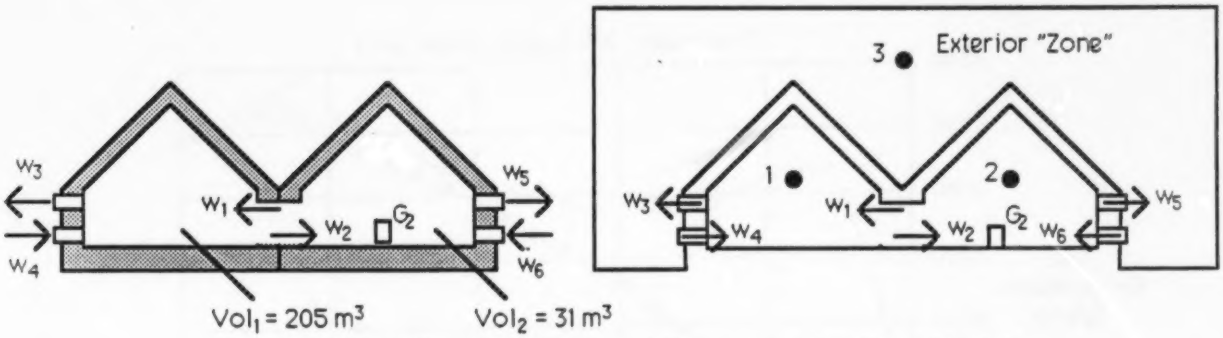


Fig. 9.8 A Two Zone Building and Corresponding Flow Model

It will be assumed that infiltration will be equal to exfiltration for each zone (i.e., $w_3 = w_4$ and $w_5 = w_6$) given by the product of the reported whole-house infiltration rate (0.35 ACH) and the respective volumetric masses. The average indoor air temperature of 16 °C will be used to compute volumetric mass quantities and mass flow rates from the reported values (i.e., a constant density of 1.22 kg/m^3 is assumed for air).

The "inter-room" mass flow rate time histories (i.e., $w_1(t)$ or equivalently $w_2(t)$), based on the reported volumetric flow rate histories, are plotted below along with the computed variation of CO_2 concentration in each zone, figures 9.9 and 9.10.

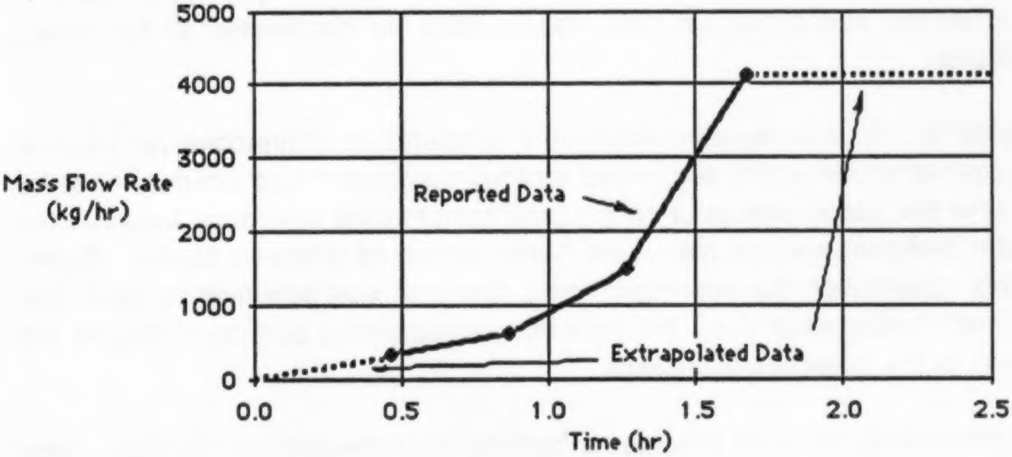


Fig. 9.9 Two Zone Example: Inter-Room Mass Flow Rate

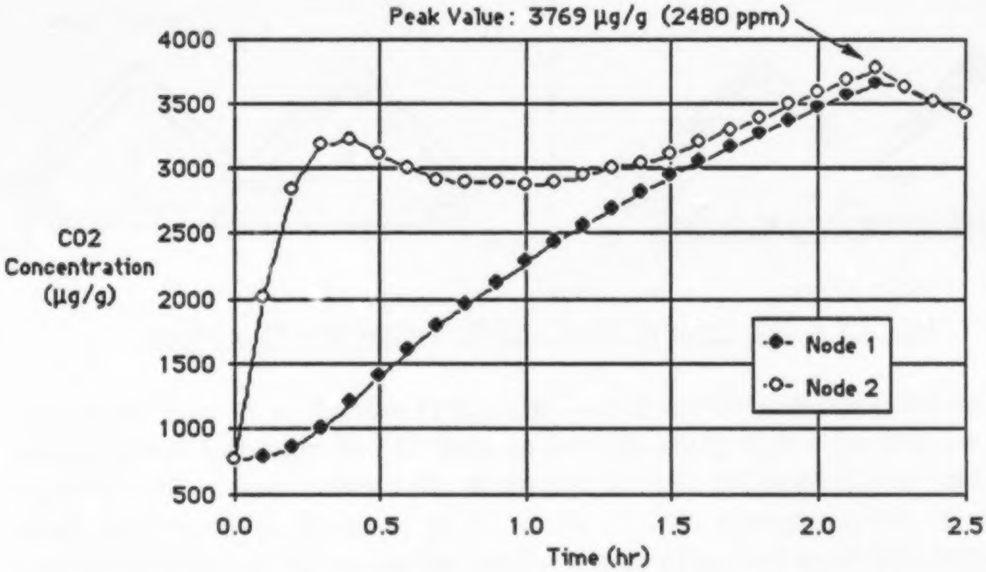


Fig. 9.10 Two Zone Example: Response Based on Measured Flow and CO_2 Generation Data

The peak CO_2 concentration measured during the test was 3709 µg/g (2440 ppm) that compares very well with the predicted concentration of 3769 µg/g (2480 ppm). It should be noted, however, that the reported flow rates were determined to an accuracy of only ± 33 % so the close agreement of experimental and analytical peak values must be considered to be largely fortuitous.

Traynor et. al. also reported inter-room temperature differences for the test considered above which suggested thermal equilibrium had been achieve by the time the heater was shut off (i.e., the temperature difference between the master bedroom and the rest of the house remained relatively steady. Based on this observation the inter-room mass flow rate was assumed to have also reached steady state (i.e., the rightmost extrapolated portion of Figure 9.9 above) for the purposes of analysis.

It is interesting, then, to consider a hypothetical extension of this test - How would CO_2 concentration vary under these (apparently) steady conditions? To answer this question an additional analysis was computed using the flow time history reported above (Figure 9.9), with flow assumed constant after 1.7 hours, and a constant generation rate (i.e., without shutting off the heater). The results of this study are plotted below. The program CONTAM, in this instance, was used to estimate both the steady state and the dynamic response of the system.

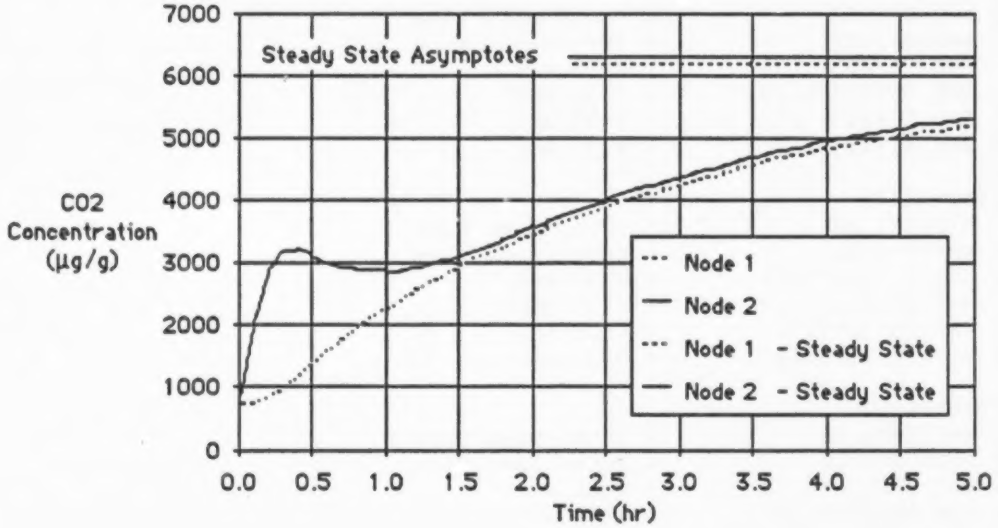


Fig. 9.11 Two Zone Example: Hypothetical Constant CO_2 Generation Rate Response

Command/data Input File

The CONTAM command/data file used for the first study is listed below. It should be noted that a large number was used for the volumetric mass of the exterior "zone" to affect a model of a practically infinite contaminant sink.

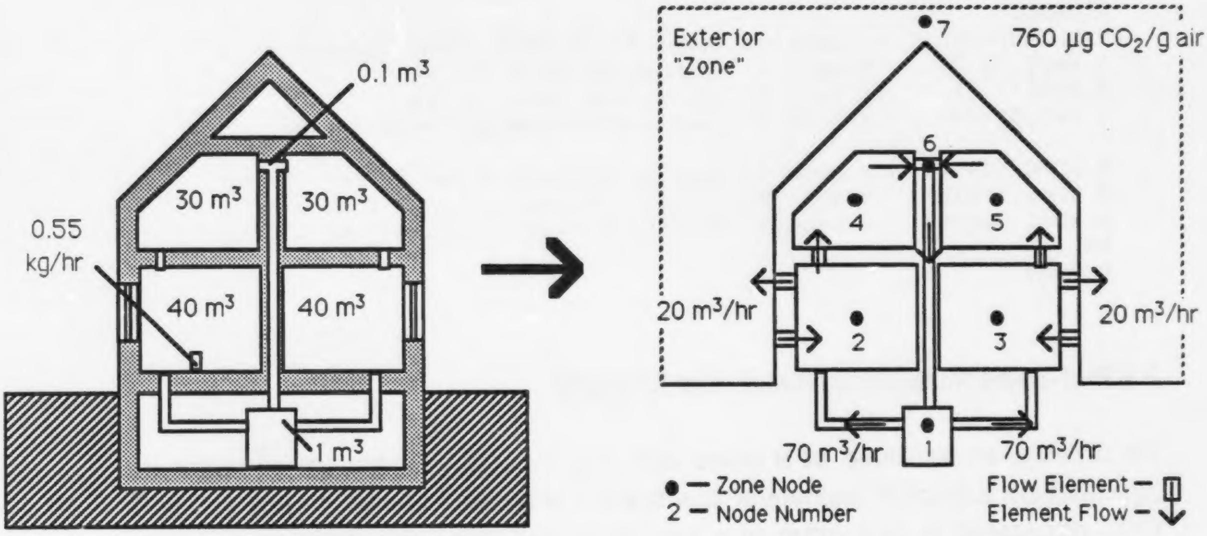
```
FLOWSYS N=3         < Two-Zone (3-Node) Example
3 BC=C              < Exterior "Zone" (Node 3) Will Have Conc. Specified
END
FLOWELEM
1 I=2,1             < Flow Element 1
2 I=1,2             < Flow Element 2
3 I=1,3             < Flow Element 3
4 I=3,1             < Flow Element 4
5 I=2,3             < Flow Element 5
6 I=3,2             < Flow Element 6
END
FLOWDAT        T=0,180/60,0.1     < Element Mass Flow Rates [=] kg/hr
TIME=0
1 W=0                      < Inter-Room Flow
2 W=0                      < Inter-Room Flow
```

```
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
<
TIME=28/60
1 W=250*1.22             < Inter-Room Flow
2 W=250*1.22             < Inter-Room Flow
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
<
TIME=52/60
1 W=500*1.22             < Inter-Room Flow
2 W=500*1.22             < Inter-Room Flow
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
<
TIME=76/60
1 W=1205*1.22            < Inter-Room Flow
2 W=1205*1.22            < Inter-Room Flow
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
<
TIME=101/60
1 W=3375*1.22            < Inter-Room Flow
2 W=3375*1.22            < Inter-Room Flow
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
<
TIME=210/60
1 W=3375*1.22            < Inter-Room Flow
2 W=3375*1.22            < Inter-Room Flow
3 W=0.35*205*1.22        < 0.35 ACH
4 W=0.35*205*1.22        < 0.35 ACH
5 W=0.35*31*1.22         < 0.35 ACH
6 W=0.35*31*1.22         < 0.35 ACH
END
EXCITDAT         < Nodal Excitation
TIME=0.0
2 CG=0.549       < Node 2: Generation Rate [=] kg/hr
3 CG=0.000760    < Node 3: Exterior CO2 Concentration [=] kg CO2/kg
<
TIME=133/60      < Kerosene heater turned off at 133 minutes.
2 CG=0.0         < Node 2: Generation Rate [=] kg/hr
3 CG=0.000760    < Node 3: Exterior CO2 Concentration [=] kg CO2/kg
<
TIME=210/60
2 CG=0.0         < Node 2: Generation Rate [=] kg/hr
```

```
3 CG=0.000760     < Node 3: Exterior CO2 Concentration [=] kg CO2/kg
<
END
DYNAMIC
T=0,150/60,0.1    < Initial Time, Final Time, Time Increment
1 V=205*1.22      < Node 1: Volumetric Mass [=] kg
2 V=31*1.22       < Node 2: Volumetric Mass [=] kg
3 V=1.0E+09       < Node 3: Exterior Volumetric Mass [=] kg
<
1 IC=0.000760     < Node 1: Initial Concentration [=] kg CO2/kg
2 IC=0.000760     < Node 2: Initial Concentration [=] kg CO2/kg
3 IC=0.000760     < Node 3: Initial Concentration [=] kg CO2/kg
END
RETURN
```

9.7 Full-Scale Multi-zone Residential Example

To provide an example of a more complex multi-zone problem consider the hypothetical full-scale residential flow system illustrated below. In this example, CO_2 generated in one room of a two story four room residence is dispersed throughout the building by the hot-air system and diluted by outside air infiltration at the rate of 0.5 ACH in the two lower rooms. The CO_2 is generated by a portable kerosene heater, whose generation characteristics are assumed to be the same as that used above in the single zone examples, is operated for 133 minutes and then turned off. The results of the analysis are plotted below illustrating the detailed dynamic variation of pollutant concentration in the building air flow system.


0.1 m³
30 m³
30 m³
0.55 kg/hr
40 m³
40 m³
1 m³
Exterior "Zone"
7
760 µg CO₂/g air
6
4
5
20 m³/hr
2
3
20 m³/hr
70 m³/hr
1
70 m³/hr
● — Zone Node
2 — Node Number
Flow Element —
Element Flow —

Actual Building

Air-Flow System Idealization

Fig. 9.12 Full-Scale Residence and Corresponding Flow Model

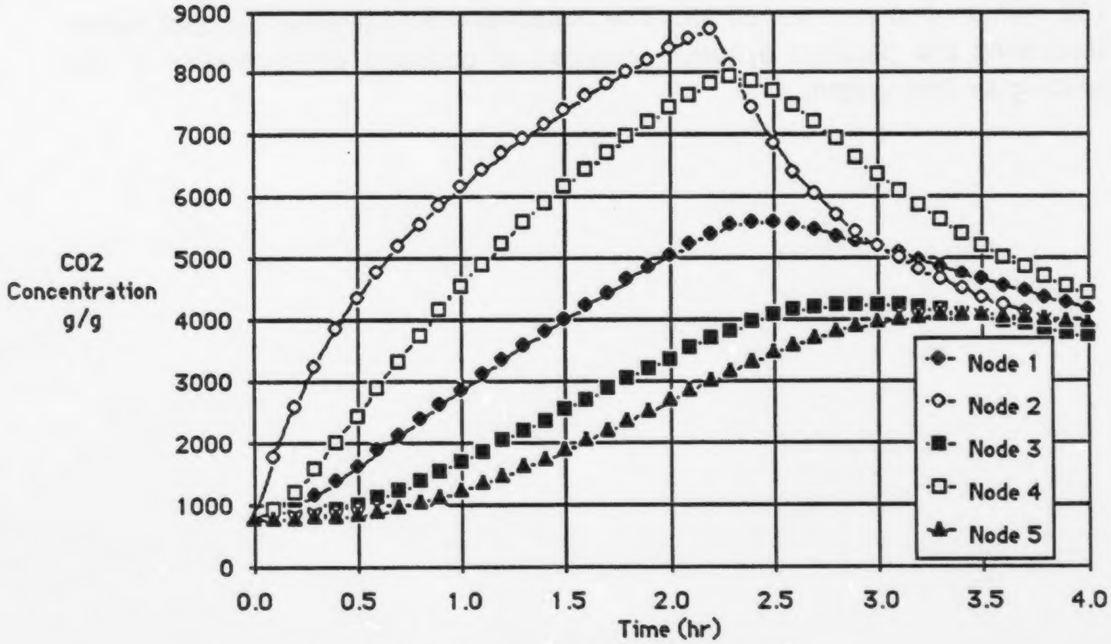


Fig. 9.13 Residential Example Response Results

Command/data Input File

The CONTAM command/data input file used for this study is listed below.

```
FLOWSYS N=7       < Six-Zone (7-node) Example
7 BC=C            < Exterior "Zone" (Node 7) Will Have Conc. Specified
END
FLOWELEM
1 I=1,2           < Flow Element 1
2 I=1,3           < Flow Element 2
3 I=7,2           < Flow Element 3
4 I=2,7           < Flow Element 4
5 I=7,3           < Flow Element 5
6 I=3,7           < Flow Element 6
7 I=2,4           < Flow Element 7
8 I=3,5           < Flow Element 8
9 I=4,6           < Flow Element 9
10 I=5,6          < Flow Element 10
11 I=6,1          < Flow Element 11
END
TIMECONS
1,2  W=70*1.2     < 0.50 Building ACH each
3,6  W=20*1.2     < 0.25 Room ACH each
7,10 W=70*1.2     < 0.50 Building ACH each
11 W=140*1.2      < 1.00 Building ACH
<
1    V=1.2*1.0    < Node 1: Volumetric Mass [=] kg
2,3  V=1.2*40.0   < Nodes 2 & 3: Volumetric Mass [=] kg
4,5  V=1.2*30.0   < Nodes 4 & 5: Volumetric Mass [=] kg
6    V=1.2*0.1    < Node 6: Volumetric Mass [=] kg
7 V=1.2*1.0E+06   < Node 7: Exterior Volumetric Mass [=] kg
END
FLOWDAT           < Element Mass Flow Rates [=] kgm/hr
TIME=0
1,2  W=70*1.2     < 0.50 Building ACH each
3,6  W=20*1.2     < 0.25 Room ACH each
7,10 W=70*1.2     < 0.50 Building ACH each
11 W=140*1.2      < 1.00 Building ACH
<
TIME=5
1,2  W=70*1.2     < 0.50 Building ACH each
3,6  W=20*1.2     < 0.25 Room ACH each
7,10 W=70*1.2     < 0.50 Building ACH each
11 W=140*1.2      < 1.00 Building ACH
END
EXCITDAT          < Nodal Excitation
TIME=0
2 CG=0.549        < Node 2: Generation Rate [=] kg/hr
7 CG=0.000760     < Node 7: Exterior CO2 Concentration [=] kg CO2/kg
<
TIME=133/60       < Kerosene Heater Turned Off at 133 minutes
2 CG=0.0          < Node 2: Generation Rate [=] kg/hr
7 CG=0.000760     < Node 7: Exterior CO2 Concentration [=] kg CO2/kg
```

```
<
TIME=5
2 CG=0.0          < Node 2: Generation Rate [=] kg/hr
3 CG=0.000760     < Node 3: Exterior CO2 Concentration [=] kg CO2/kg
<
END
DYNAMIC
T=0,4,0.5         < Initial Time, Final Time, Time Increment
1    V=1.2*1.0    < Node 1: Volumetric Mass [=] kg
2,3  V=1.2*40.0   < Nodes 2 & 3: Volumetric Mass [=] kg
4,5  V=1.2*30.0   < Nodes 4 & 5: Volumetric Mass [=] kg
6    V=1.2*0.1    < Node 6: Volumetric Mass [=] kg
7 V=1.2*1.0E+09   < Node 7: Exterior Volumetric Mass [=] kg
<
1,7 IC=0.000760   < Initial Concentration [=] kg CO2/kg
END
RETURN
```

It will be noticed that, in this case, system time constants were to be computed. The results of the time constants analysis are listed below;

```
==== TIMECONS: TIME CONSTANTS - CONTAMINANT DISPERSAL SYSTEM

     Convergence parameter, epsilon, ... 0.100E-15

  == Element Mass Flow Rates

     Elem    Value    Elem    Value    Elem    Value    Elem    Value    Elem    Value
        1    84.0        2    84.0        3    24.0        4    24.0        5    24.0
        6    24.0        7    84.0        8    84.0        9    84.0       10    84.0
       11    168.

  == Net Total Mass Flow

             "*" = independent DOFs          "U" = undefined DOFs.

     Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
        1    .000        2    .000        3    .000        4    .000        5    .000
        6    .000       7*    .000

  == Nodal Volumetric Mass

             "*" = independent DOFs          "U" = undefined DOFs.

     Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
        1    1.20        2    48.0        3    48.0        4    36.0        5    36.0
        6    .120       7*  0.120E+07

  == Nominal Time Constants

     Node    Value    Node    Value    Node    Value    Node    Value    Node    Value
        1  0.714E-02  2     .444        3    .444        4    .429        5    .429
        6  0.714E-03  7   0.250E+05

  == Actual Time Constants
```

```
Num.    Value    Num.    Value     Num.    Value    Num.   Value     Num.    Value
  1   0.714E-03   2   0.714E-02   3     .230      4     .429       5     .444
  6    3.73       7  -0.852E+16

Number of iterations used ...   11
```

PART II References

[1] Wilson, E. L. & Hoit, M. I., "A Computer Adaptive Language for the Development of Structural Analysis Programs," Computers & Structures, Vol. 19, No. 3, pp 321-338, 1984

[2] Eberlein, P.J. & Boothroyd, J., "Contribution II/12: Solution to the Eigenproblem by a Norm Reducing Jacobi Type Method," Handbook for Automatic Computation: Volume II: Linear Algebra, Wilkinson, J.H. & Reinsch,& Reinsch, C. - editors, Springer-Verlag, 1971

[3] Traynor, G.W., Allen, J.R., Apte, M.G., Girman, J.R., & Hollowell, C.D., "Pollution Emissions from Portable Kerosene-Fired Space Heaters", Environmental Science & Technology, Vol. 17, June 1983, pp.369-371

[4] Traynor, G.W., Apte, M.G., Carruthers, A.R., Dillworth, J.F., Grimsrud, D.T., & Thompson, W.T., "Indoor Air Pollution and Inter-Room Pollutant Transport Due to Unvented Kerosene-Fired Space Heaters,", Lawrence Berkeley Laboratory - University of California, Applied Science Division, LBL-17600, Feb., 1984

Appendix - FORTRAN 77 Source Code

The program CONTAM86 is listed below. In this listing you will note that compiler directives to "include"code stored in separate "include files" are used. These "include files" contain common block data specifications that are shared by many subroutines. The contents of these include files are listed on the last page of this appendix.

```
C--------------------------------------------------------------CONTAM86
      PROGRAM CONTAM
C----------------------------------------------------------------------
C--PRO:CONTAM - BUILDING CONTAMINANT DISPERSAL ANALYSIS PROGRAM
C              VERSION FY86
C
C----- Developed by JAMES AXLEY
C                    Dept. of Architecture, Cornell University
C                    Building Environment Division, NBS
C                    Fall, 1986
C       Using;
C       A) CAL-SAP Library of subroutines developed by ED WILSON,
C          U.C. BERKELEY
C       B) MicroSoft FORTRAN V2.2 Compiler for Apple Macintosh
C          For Mac
C            1. Set logical unit numbers, in SUBROUTINE INITIO, as;
C                  NTR = 9  ;  NTW = 9   ;   NCMD = 9
C            2. INCLUDE statements use <filename>.INC (i.e., without ')
C            3. In SUBROUTINE PROMPT use: WRITE(NTW,'(A,\)') STRING
C       C) IBM PC Professional FORTRAN (Ryan-McFarland)
C            1. Set logical unit numbers, in SUBROUTINE INITIO, as;
C                  NTR = 5  ;  NTW = 6   ;   NCMD = 5
C            2. INCLUDE statements use '<filename>.INC' (i.e., with ')
C            3. In SUBROUTINE PROMPT use: WRITE(NTW,'(A)') STRING
C
C       Memory for dynamically allocated/defined arrays is located in
C       vector IA(MTOT) in blank common.  To increase or decrease this
C       area alter the dimension of IA, in the section 0.0 below, set
C       MTOT, in section 1.0 below, equal to this new dimension, and
C       recompile the code.  As integers are 4 bytes wide, memory
C       dedicated to IA(MTOT) is equal to MTOT*4 bytes.
C----------------------------------------------------------------------
C
      IMPLICIT REAL*8 (A-H,O-Z)

C----------------------------------------------------------------------
C--0.0 DATA SPECIFICATIONS & COMMON STORAGE
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(20000)

      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC
      INCLUDE CMDCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR

C----------------------------------------------------------------------
C--1.0 INITIALIZE INTERNAL VARIABLES
C----------------------------------------------------------------------

      MTOT = 20000
      CALL INITAR(MTOT)
      CALL INITIO
      CALL INITCN
      ERR = .FALSE.

C----------------------------------------------------------------------
C--2.0 WRITE BANNER
C----------------------------------------------------------------------
      CALL BANNER(NTW)
      CALL BANNER(NOT)
      WRITE(NOT,2200) (FNAME(1:LFNAME)//'.OUT')
 2200 FORMAT(/' ---- RESULTS OUTPUT FILE: ',(A))
```

```
C----------------------------------------------------------------------
C--3.0 COMMAND PROCESSOR LOOP
C----------------------------------------------------------------------
C
C---3.1 CHECK BLANK COMMON STORAGE
C
   30 NSTOR = (IDIR-NEXT-20)*IP(1)/IP(2)
      IF(NSTOR.LE.100) WRITE(NTW,2300) NSTOR
 2300 FORMAT(
     +' **** WARNING: Array storage available =',I9,' real numbers.')
C
C---3.2 GET COMMAND LINE
C
      IF(MODE.EQ.'INTER') CALL PROMPT(' CMND>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
C
C---3.3 INTERPRET COMMAND LINE
C
C----- GET COMMAND & ARRAY NAMES, IF ANY
C
      CALL FREEC(' ',NCMND,8,1)
      CALL FREEC('A',M1(1),4,7)
C
C----- INTRINSIC COMMANDS
C
      IF((NNCMND.EQ.'H').OR.(NNCMND.EQ.'HELP')) THEN
        IF(MODE.EQ.'BATCH') THEN
          WRITE(NTW,2310)
          WRITE(NOT,2310)
          CALL RETRN
        ELSE
          CALL HELP
        ENDIF

      ELSEIF(NNCMND.EQ.'ECHO-ON') THEN
        ECHO = .TRUE.

      ELSEIF(NNCMND.EQ.'ECHO-OFF') THEN
        ECHO = .FALSE.

      ELSEIF((NNCMND.EQ.'L').OR.(NNCMND.EQ.'LIST')) THEN
        IF(MODE.EQ.'BATCH') THEN
          WRITE(NTW,2310)
          WRITE(NOT,2310)
          CALL RETRN
        ELSE
          CALL LIST
        ENDIF

      ELSEIF((NNCMND.EQ.'P').OR.(NNCMND.EQ.'PRINT')) THEN
        CALL PRINT

      ELSEIF((NNCMND.EQ.'D').OR.(NNCMND.EQ.'DIAGRAM')) THEN
        CALL DIAGRM

      ELSEIF(NNCMND.EQ.'SUBMIT') THEN
        IF(MODE.EQ.'BATCH') THEN
          WRITE(NTW,2310)
          WRITE(NOT,2310)
          CALL RETRN
        ELSE
          CALL SUBMIT
        ENDIF

      ELSEIF(NNCMND.EQ.'RETURN') THEN
        IF(MODE.EQ.'INTER') THEN
          WRITE(NTW,2320)
        ELSE
          CALL RETRN
        ENDIF

      ELSEIF((NNCMND.EQ.'Q').OR.(NNCMND.EQ.'QUIT')) THEN
        STOP
C
C----- CONTAM COMMANDS
C
      ELSEIF(NNCMND.EQ.'FLOWSYS') THEN
```

```
          CALL FLOSYS

        ELSEIF(NNCMND.EQ.'FLOWELEM') THEN
          CALL FLOELM

        ELSEIF(NNCMND.EQ.'STEADY') THEN
          CALL STEADY

        ELSEIF(NNCMND.EQ.'TIMECONS') THEN
          CALL TIMCON

        ELSEIF(NNCMND.EQ.'FLOWDAT') THEN
          CALL FLODAT

        ELSEIF(NNCMND.EQ.'EXCITDAT') THEN
          CALL EXCDAT

        ELSEIF(NNCMND.EQ.'DYNAMIC') THEN
          CALL DYNAM

        ELSEIF(NNCMND.EQ.'RESET') THEN
          CALL RESET

        ELSE
          WRITE(NTW,2330)
          IF(MODE.EQ.'BATCH') THEN
            CALL RETRN
          ENDIF

        ENDIF
        GO TO 30
C
 2310 FORMAT(' **** ERROR: Command not defined in BATCH mode.')
 2320 FORMAT(' **** ERROR: Command not defined in INTERACTIVE mode.')
 2330 FORMAT(' **** ERROR: Command not defined.')

      END

C------------------------------------------------------------------INITAR
      SUBROUTINE INITAR(MTOT)
C--SUB:INITAR - INITIALIZES DYNAMIC ARRAY MANAGER VARIABLES
C               IN BLANK COMMON AND LABELED COMMON /ARYCOM/

      INCLUDE ARYCOM.INC

      NUMA = 0
      NEXT = 1
      IDIR = MTOT
      IP(1) = 4
      IP(2) = 8
      IP(3) = 1
      RETURN
      END

C------------------------------------------------------------------INITIO
      SUBROUTINE INITIO
C--SUB:INITIO - INITIALIZES LABELED COMMON /IOCOM/
C               OPENS DEFAULT RESULTS OUTPUT FILE

      INCLUDE IOCOM.INC
      LOGICAL FOUND

      NTR = 9
      NTW = 9
      NCMD = 9
      NIN = 10
      NOT = 11
      ND1 = 12
      ND2 = 13
      ND3 = 14
      ND4 = 15
      FNAME = 'CONTAM'
      LFNAME = 6
      EXT = '   '
      CALL NOPEN(NOT,(FNAME(1:LFNAME)//'.OUT'),'FORMATTED')
      MODE = 'INTER'
      ECHO = .TRUE.
      RETURN
      END
```

```
C------------------------------------------------------------------INITCN
      SUBROUTINE INITCN
C--SUB:INITCN - INITIALIZES CONTAM LABELED COMMON /CNTCOM/

      INCLUDE CNTCOM86.INC

      NFNOD = 0
      NFEQN = 0
      NFBAN = 0
      NFELM = 0
      MPV   = 0
      MPF   = 0
      MPC   = 0
      MPG   = 0
      MPREQ = 0
      EP    = 1.0D-16
      RETURN
      END

C------------------------------------------------------------------BANNER
      SUBROUTINE BANNER(LUN)
C--SUB: BANNER - WRITES PROGRAM BANNER TO LOGICAL UNIT LUN

      COMMON MTOT,NP,IA(1)

      WRITE(LUN,2000) MTOT
 2000 FORMAT(//,1X,78(1H-),/,
     .' |                        C O N T A M 8 6',T79,'|',/,
     .' |          Contaminant Dispersal Analysis for Building Systems'
     .,T79,'|',/,
     .' |                 Version FY86 - Jim Axley - Cornell & NBS',
     .T79,'|',/,1X,78(1H-),/,65X,'MTOT:',I9)

      RETURN
      END

C-----------------------------------------------------------------------C
C                                                                       C
C               I N T R I N S I C   C O M M A N D S                     C
C                                                                       C
C-----------------------------------------------------------------------C

C====================================================================HELP
      SUBROUTINE HELP
C--SUB: HELP - PROVIDES ON-SCREEN HELP
C
C--HELP LIST------------------------------------------------------------
C
C   .' HELP (H)              List available commands.',/,
C-----------------------------------------------------------------------

      INCLUDE IOCOM.INC

      WRITE(NTW,2000)
      PAUSE '   — Enter <CR> to continue.'
      WRITE(NTW,2010)
      PAUSE '   — Enter <CR> to continue.'
      WRITE(NTW,2020)
      PAUSE '   — Enter <CR> to continue.'
      WRITE(NTW,2030)
      PAUSE '   — Enter <CR> to continue.'
      WRITE(NTW,2040)

      RETURN
C
C ---------------------- HELP LISTS -------------------------------------
C
 2000 FORMAT(/,' ——— INTRINSIC COMMANDS',//,
     .' HELP (H)                List available commands.',/,
     .' ECHO-ON                 Echo results to screen.',/,
     .' ECHO-OFF                Do not echo results to screen.',/,
     .' LIST (L)                List the directory of all arrays.',/,
     .' PRINT (P)  A=<array>    Print array named <array>.',/,
     .' DIAGRAM (D) A=<array>   Diagram array named M1.',/,
     .' SUBMIT F=<filename>     Read commands from  batch <filename>.',/,
     .' RETURN                  Last command in batch  <filename>.',/,
     .' QUIT (Q)                Quit program.'/)
C
 2010 FORMAT(/,' ——— CONTAM COMMANDS',//,
     .' FLOWSYS N=n1           Flowsystem control variables.',/,
     .' n2,n3,n4 BC=c1          n1 = number of flow nodes',/,
     .' ...                     n2,n3,n4 = node: first, last, incr.',/,
     .' END                     c1 = boundary condition; G or C',//,
     .' FLOWELEM               Flow element command/data group.',/,
     .' n1 I=n2,n3 E=n4         n1 = element number',/,
     .'  ...                    n2,n3 = element end nodes',/,
     .' END                     n4 = filter efficiency',//,
     .' STEADY                 Steady state solution.',/,
     .' n1,n2,n3 W=n4           n1,n2,n3 = elem: first, last, incr.',/,
     .' ...                     n4 = element flow rate',/,
     .' n5,n6,n7 CG=n8          n5,n6,n7 = node: first, last, incr.',/,
     .' ...                     n8 = prescribed conc. or gen. rate',/,
     .' END')
 2020 FORMAT(/,
     .' TIMECONS E=n1          Time constant solution, n1 = epsilon',/,
     .' n2,n3,n4 W=n5           n2,n3,n4 = elem: first, last, incr.',/,
     .' ...                     n5 = element flow rate',/,
     .' : ',/,
     .' n6,n7,n8 V=n9           n6,n7,n8 = node: first, last, incr.',/,
     .' ...                     n9 = nodal volumetric mass',/,
     .' END')
 2030 FORMAT(/,
     .' FLOWDAT [T=n1,n2,n3]  Generate element flow time histories.',/,
     .' TIME=n1                 n1 = time',/,
     .' n1,n2,n3 W=n4           n1,n2,n3 = elem: first, last, incr.',/,
     .' ...                     n4 = element mass flow rate.',/,
     .' :',/,
     .' END',//,
     .' EXCITDAT [T=n1,n2,n3] Generate excitation time histories.',/,
     .' TIME=n1                 n1 = time',/,
     .' n1,n2,n3 CG=n4           n1,n2,n3 = node: first, last, incr.',/,
     .' ...                     n4 = excitation: conc. or gen. rate.',/,
     .' :',/,
     .' END')
 2040 FORMAT(/,
     .' DYNAMIC                Dynamic solution.',/,
     .' T=n1,n2,n3 [A=n4] [PI=n5] [PS=n6]',/,
     .' n7,n8,n9 V=n10          n1,n2,n3 = init,final,incr; n4 =alpha',/,
     .' ...                     n5 = print interval; n6 = plot scale',/,
     .' :                       n7,n8,n9 = node: first, last, incr.',/,
     .' n7,n8,n9 IC=n11         n10 = nodal volumetric mass',/,
     .' ...                     n11 = initial nodal concentration',/,
     .' END ',//,
     .' RESET                  Reset CONTAM for new problem.')

      END

C------------------------------------------------------------------LIST
      SUBROUTINE LIST
C--SUB:LIST - LIST DIRECTORY OF ALL ARRAYS IN BLANK COMMON
C
C--HELP LIST-----------------------------------------------------------
C
C    .' LIST (L)              List the directory of all arrays.',/,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC

      CHARACTER*1 NAM(4),LOC(4,2),TYPE(9,3),STOR(13,2)
      CHARACTER*1 CHK
C
      DATA TYPE/'I','N','T','E','G','E','R',' ',' ',
     1          'R','E','A','L',' ',' ',' ',' ',' ',
     2          'C','H','A','R','A','C','T','E','R'/
C
      DATA LOC/'C','O','R','E','D','I','S','K'/
C
      DATA STOR/'S','E','Q','U','E','N','T','I','A','L',' ',' ',' ',
     1          'D','I','R','E','C','T',' ','A','C','C','E','S','S'/
C
C-----LIST DIRECTORY OF ALL ARRAYS IN DATA BASE
      IF(NUMA.EQ.0) GO TO 900
C
C-----WRITE HEADER FOR SCREEN LISTING OF FILE DATA
      WRITE(NTW,1000)
C
C-----START COUNT OF LINES TO SCREEN
      IL = 5
C
      IC = IDIR
      DO 100 I=1,NUMA
      IL = IL + 1
      ILOC = 1
      IST = 0
      IA6 = IA(IC+6)
      IA7 = IA(IC+7)
      IA9 = IA(IC+9)
C-----CHECK FOR LOCATION AND STORAGE TYPE
      IF(IA9.GT.0) ILOC=2
      IF(IA7.LT.0) ILOC=2
      IF(IA7.EQ.-1) IST=1
      IF(IA7.EQ.-2) IST=2
      IF(IA9.GT.0) IST=3
      IPN = IC - 1
      DO 10 J=1,4
      IPN = IPN + 1
   10 NAM(J) = CHAR(IA(IPN))
C-----WRITE DATA TO TERMINAL
      IF(IST.EQ.0) WRITE(NTW,1100) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),(TYPE(K,IA6),K=1,9),
     * (LOC(L,ILOC),L=1,4)
C
      IF(IST.EQ.1) WRITE(NTW,1100) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),(TYPE(K,IA6),K=1,9),
     * (LOC(L,ILOC),L=1,4),(STOR(M,1),M=1,13)
C
      IF(IST.EQ.2) WRITE(NTW,1300) (NAM(J),J=1,4),
     * IA(IC+4),(LOC(L,ILOC),L=1,4),(STOR(M,2),M=1,13)
C
      IF(IST.EQ.3) WRITE(NTW,1200) (NAM(J),J=1,4),
     * IA(IC+4),IA(IC+5),IA(IC+6),(LOC(L,ILOC),L=1,4),
     * (STOR(M,2),M=1,13)
C
      IC = IC + 10
C-----CHECK FOR NUMBER OF LINES PRINTED
      IF(IL.LT.20) GO TO 100
      IF(I.EQ.NUMA) GO TO 100
      CALL PROMPT('   ** Do you want more ? (Y/N) ')
      READ(NTR,2200)
      IF((CHK.EQ.'n').OR.(CHK.EQ.'N')) GO TO 900
      IL = 0
      WRITE(NTW,2000)
  100 CONTINUE
C
  900 RETURN
C
 1000 FORMAT(' ---- LIST: ARRAY LIST',//,
     * ' Name',2X,'Number',2X,'Number',5X,'Data',5X,
     * 'Location',5X,'Storage',/,8X,'Rows',2X,
     * 'Columns',5X,'Type',19X,'Type',/)
 1100 FORMAT(1X,4A1,2X,I4,4X,I4,5X,9A1,4X,4A1,4X,13A1)
 1200 FORMAT(1X,4A1,'  NI=',I4,'  NR=',I4,'  NC=',I4,5X,4A1,4X,13A1)
 1300 FORMAT(1X,4A1,3X,'RECORD LENGTH = ',I6,7X,4A1,4X,13A1)
 2000 FORMAT()
 2200 FORMAT(1A1)
      END

C-----------------------------------------------------------------PRINT
      SUBROUTINE PRINT
C--SUB:PRINT - COMMAND TO "PRINT" ARRAY TO RESULTS OUTPUT FILE
C
C--HELP LIST-----------------------------------------------------------
C
C    .' PRINT (P)  A=<array>  Print array named <array>.',/,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC
      INCLUDE CMDCOM.INC

C-----PRINT OF REAL OR INTEGER ARRAY
      CALL PROMS(1)
C-----LOCATE MATRIX TO BE PRINTED
      IF(ECHO) WRITE(NTW,2000) M1
      WRITE(NOT,2000) M1
      CALL LOCATE(M1,NA,NR,NC)
      IF(NA.EQ.0) THEN
        WRITE(NTW,2010) M1
        WRITE(NOT,2010) M1
        CALL ABORT
        RETURN
      ELSEIF(NA.LT.0) THEN
        WRITE(NTW,2020) M1
        WRITE(NOT,2020) M1
        CALL ABORT
        RETURN
      ELSE
        IF(NP.EQ.1) CALL IPRT(IA(NA),NR,NC)
        IF(NP.EQ.2) CALL RPRT(IA(NA),NR,NC)
      ENDIF

      RETURN
C
 2000 FORMAT(/' ——— PRINT  OF  ARRAY  "',4A1,'"')
 2010 FORMAT(' **** ERROR: Array "',4A1,'" does not exist.')
 2020 FORMAT(' **** ERROR: Array "',4A1,'" is out of core.')
      END

C------------------------------------------------------------------- IPRT
      SUBROUTINE IPRT(N,NR,NC)
C--SUB: IPRT - PRINTS INTEGER ARRAY TO RESULTS OUTPUT FILE

      DIMENSION N(NR,NC)

      INCLUDE IOCOM.INC

      NUMC = 14
      DO 100 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000) (K,K=I,IN)
      IF(ECHO) WRITE(NTW,2000) (K,K=I,IN)
      DO 100 J=1,NR
      WRITE(NOT,2001) J,(N(J,K),K=I,IN)
      IF(ECHO) WRITE(NTW,2001) J,(N(J,K),K=I,IN)
  100 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',14I5)
 2001 FORMAT(' ROW',I4,14I5)
      END

C------------------------------------------------------------------- RPRT
      SUBROUTINE RPRT(A,NR,NC)
C--SUB: RPRT - PRINTS REAL ARRAY TO RESULTS OUTPUT FILE

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NR,NC)

      INCLUDE IOCOM.INC

      XMAX = 0.00
      DO 50 I=1,NR
      DO 50 J=1,NC
      XX = DABS(A(I,J))
      IF(XX.GT.XMAX) XMAX = XX
   50 CONTINUE
      M = 1
      IF(XMAX.LT.99999.) M = 2
      IF(XMAX.LT.0.1000) M = 1
      IF(XMAX.EQ.0.0)    M = 2
C
      NUMC = 6
      DO 100 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000) (K,K=I,IN)
      IF(ECHO) WRITE(NTW,2000) (K,K=I,IN)
      DO 100 J=1,NR
      IF(M.EQ.1) THEN
        WRITE(NOT,2001) J,(A(J,K),K=I,IN)
        IF(ECHO) WRITE(NTW,2001) J,(A(J,K),K=I,IN)
      ELSEIF(M.EQ.2) THEN
```

```
        WRITE(NOT,2002) J,(A(J,K),K=I,IN)
        IF(ECHO) WRITE(NTW,2002) J,(A(J,K),K=I,IN)
      ENDIF
  100 CONTINUE
C     RETURN

 2000 FORMAT(/' COL# =',6I12)
 2001 FORMAT(' ROW',I4,6E12.5)
 2002 FORMAT(' ROW',I4,6F12.5)
      END

C-----------------------------------------------------------------DIAGRM
      SUBROUTINE DIAGRM
C--SUB:DIAGRM - COMMAND TO "DIAGRAM" ARRAY TO RESULTS OUTPUT FILE
C
C--HELP LIST-------------------------------------------------------------
C
C    .' DIAGRAM (D) A=<array> Diagram array named M1.',/,
C-----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CMDCOM.INC

C-----PRINT OF REAL OR INTEGER ARRAY
      CALL PROMB(1)
C-----LOCATE MATRIX TO BE PRINTED
      IF(ECHO) WRITE(NTW,2000) M1
      WRITE(NOT,2000) M1
      CALL LOCATE(M1,NA,NR,NC)
      IF(NA.EQ.0) THEN
        WRITE(NTW,2010) M1
        WRITE(NOT,2010) M1
        CALL ABORT
        RETURN
      ELSEIF(NA.LT.0) THEN
        WRITE(NTW,2020) M1
        WRITE(NOT,2020) M1
        CALL ABORT
        RETURN
      ELSE
        IF(NP.EQ.1) CALL IDIAGR(IA(NA),NR,NC)
        IF(NP.EQ.2) CALL RDIAGR(IA(NA),NR,NC)
      ENDIF

      RETURN
C
 2000 FORMAT(/' ——— DIAGRAM  OF  ARRAY  "',4A1,'"')
 2010 FORMAT(' **** ERROR: Array "',4A1,'" does not exist.')
 2020 FORMAT(' **** ERROR: Array "',4A1,'" is out of core.')
      END

C-----------------------------------------------------------------IDIAGR
      SUBROUTINE IDIAGR(N,NR,NC)
C--SUB: IDIAGR - "DIAGRAMS" INTEGER ARRAY TO RESULTS OUTPUT FILE

      INTEGER N(NR,NC)
      CHARACTER*1 ICON(36)

      INCLUDE IOCOM.INC

C-----DIAGRAM INTEGER ARRAY
      NUMC = 36
      DO 200 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000) (INT(K/10),K=I,IN)
      WRITE(NOT,2010) ((K-INT(K/10)*10),K=I,IN)
      IF(ECHO) WRITE(NTW,2000) (INT(K/10),K=I,IN)
      IF(ECHO) WRITE(NTW,2010) ((K-INT(K/10)*10),K=I,IN)
      DO 200 J=1,NR
      DO 100 K=I,IN
       ICON(K) = '*'
       IF(N(J,K).EQ.0) ICON(K) = ' '
  100 CONTINUE
       WRITE(NOT,2020) J,(ICON(K),K=I,IN)
       IF(ECHO) WRITE(NTW,2020) J,(ICON(K),K=I,IN)
  200 CONTINUE
C
```

```
      RETURN

 2000 FORMAT(/' COL# =',36(1X,I1))
 2010 FORMAT(7X,36(1X,I1))
 2020 FORMAT(' ROW',I3,36(1X,A1))
      END

C----------------------------------------------------------------RDIAGR
      SUBROUTINE RDIAGR(A,NR,NC)
C--SUB: RDIAGR - "DIAGRAMS" REAL ARRAY TO RESULTS OUTPUT FILE

      REAL*8 A(NR,NC)
      CHARACTER*1 ICON(36)

      INCLUDE IOCOM.INC

C-----DIAGRAM INTEGER ARRAY
      NUMC = 36
      DO 200 I=1,NC,NUMC
      IN = I + NUMC - 1
      IF(IN.GT.NC) IN = NC
      WRITE(NOT,2000) (INT(K/10),K=I,IN)
      WRITE(NOT,2010) ((K-INT(K/10)*10),K=I,IN)
      IF(ECHO) WRITE(NTW,2000) (INT(K/10),K=I,IN)
      IF(ECHO) WRITE(NTW,2010) ((K-INT(K/10)*10),K=I,IN)
      DO 200 J=1,NR
      DO 100 K=I,IN
       ICON(K) = '*'
       IF(A(J,K).EQ.0.0D0) ICON(K) = ' '
  100 CONTINUE
       WRITE(NOT,2020) J,(ICON(K),K=I,IN)
       IF(ECHO) WRITE(NTW,2020) J,(ICON(K),K=I,IN)
  200 CONTINUE
C
      RETURN

 2000 FORMAT(/' COL# =',36(1X,I1))
 2010 FORMAT(7X,36(1X,I1))
 2020 FORMAT(' ROW',I3,36(1X,A1))
      END

C----------------------------------------------------------------SUBMIT
       SUBROUTINE SUBMIT
C--SUB: SUBMIT - SWITCHES TO BATCH MODE AND OPENS BATCH COMMAND FILE
C
C--HELP LIST-----------------------------------------------------------
C
C    .' SUBMIT F=<filename>   Read commands from  batch <filename>.',/.
C----------------------------------------------------------------------

       INCLUDE IOCOM.INC
       LOGICAL FOUND

       CALL FREEC('F',FNAME,12,1)
       INQUIRE(FILE=FNAME(1:LENTRM(FNAME)),EXIST=FOUND)
       IF(FOUND) THEN
         MODE = 'BATCH'
         NCMD = NTN
         LFNAME = LENTRM(FNAME)
         WRITE(NTW,2010) FNAME
         WRITE(NOT,2010) FNAME
 2010    FORMAT(' **** CONTAM set to BATCH mode using file: ',A)
         OPEN(NCMD,FILE=FNAME(1:LFNAME),STATUS='OLD')
             REWIND NCMD
         CLOSE(NOT)
         CALL NOPEN(NOT,(FNAME(1:LFNAME)//'.OUT'),'FORMATTED')
         CALL BANNER(NOT)
             WRITE(NOT,2020) (FNAME(1:LFNAME)//'.OUT')
 2020    FORMAT(/' ---- RESULTS OUTPUT FILE: ',(A))

       ELSE
         WRITE(NTW,2030)
 2030    FORMAT('   -- NOTE: Submit file not found.')
         CALL ABORT
       ENDIF

       RETURN
       END
```

```
C-----------------------------------------------------------------RETRN
      SUBROUTINE RETRN
C--SUB:RETRN - RETURNS TO INTERACTIVE MODE
C
C--HELP LIST-----------------------------------------------------------
C
C    .' RETURN              Last command in batch <filename>.',/.
C----------------------------------------------------------------------

      INCLUDE IOCOM.INC

      CLOSE(NCMD)
      CLOSE(NOT)
      FNAME = 'CONTAM'
      LFNAME = 6
      OPEN(NOT,FILE=(FNAME(1:LFNAME)//'.OUT'),STATUS='OLD',
     +FORM='FORMATTED')
      REWIND NOT
   10 READ(NOT,*,END=20)
      GO TO 10
   20 BACKSPACE(NOT)
      NCMD = NTR
      MODE = 'INTER'

      WRITE(NTW,2010)
      WRITE(NOT,2010)
 2010 FORMAT(' **** CONTAM returned to INTERACTIVE mode.')

      RETURN
      END

C---------------------------------------------------------------------C
C                                                                     C
C                    C O N T A M   C O M M A N D S                    C
C                                                                     C
C---------------------------------------------------------------------C

C----------------------------------------------------------------FLOSYS
      SUBROUTINE FLOSYS
C--SUB:FLOSYS - COMMAND TO READ & PROCESS FLOW SYSTEM CONTROL VARIABLES
C               ESTABLISHES FLOW SYSTEM EQUATION NUMBERS & B.C.
C
C--HELP LIST-----------------------------------------------------------
C
C    .' FLONSYS N=n1         Flowsystem control variables.',/,
C    .' n2,n3,n4 BC=c1        n1 = number of flow nodes',/,
C    .' ...                   n2,n3,n4 = node: first, last, incr.',/,
C    .' :                     c1 = boundary condition; G or C',/,
C    .' n2,n3,n4 V=n5         n5 = nodal volumetric mass',/,
C    .' ...',/,
C    .' END',//,
C----------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      INTEGER IJK(3)
      EXTERNAL BCDATO

      ERR = .FALSE.
C
C--1.0 READ NUMBER OF FLOW SYSTEM NODES
C
      CALL FREEI('N',NFNOD,1)

      IF(NFNOD.LE.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
        ERR = .TRUE.
        GO TO 400
      ENDIF
 2100 FORMAT(' **** ERROR: Number of flow system nodes must be greater',
     +' than 0.')

      IF(MODE.EQ.'INTER') THEN
        WRITE(NTW,2110)
        WRITE(NTW,2120) NFNOD
        WRITE(NTW,2130)
      ENDIF

      WRITE(NOT,2110)
      WRITE(NOT,2120) NFNOD
      WRITE(NOT,2130)
 2110 FORMAT(/,' ——— FLOWSYS: FLOW SYSTEM CONTROL VARIABLES')
 2120 FORMAT(/
     +'       Number of flow system nodes ......',I5)
 2130 FORMAT(/,'    — Node Boundary Conditions')

      NFEQN = NFNOD
C
C--2.0 DEFINE KEQ ARRAY AND NUMBER EQUATIONS IN NODE ORDER
C
      CALL DELETE('KEQ ')
      CALL DEFINI('KEQ  ',MPKEQ,NFNOD,1)
      NN = 0
      DO 20 N=MPKEQ,MPKEQ+NFNOD-1
      NN = NN+1
   20 IA(N) = NN
C
C--3.0 PROCESS BOUNDARY CONDITION DATA
C
      CALL DATGEN(BCDATO,0,ERR)
C
C--4.0 REPORT BC IF NO ERROR ENCOUNTERED, ELSE ABORT
C
  400 IF(ERR) THEN
        CALL DELETE('KEQ ')
        MPKEQ = 0
        ERR = .FALSE.
        CALL ABORT
      ELSE
        IF(ECHO) WRITE(NTW,2400)
        WRITE(NOT,2400)
        IF(ECHO) WRITE(NTW,2410) ((N),IA(N+MPKEQ-1),N=1,NFNOD)
        WRITE(NOT,2410) ((N),IA(N+MPKEQ-1),N=1,NFNOD)
      ENDIF

      RETURN

 2400 FORMAT(/,
     .6X,'Negative Eqtn-# = concentration-prescribed boundary.',/,
     .6X,'Positive Eqtn-# = generation-prescribed boundary.',//,
     .4X,5('  Node   Eqtn',2X))
 2410 FORMAT((4X,5(I6,1X,I6,2X)))
      END

C———————————————————————————————————————————————————————BCDATO
      SUBROUTINE BCDATO(N,ERR)
C--SUB:BCDATO - READS FLOW B.C. DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
      LOGICAL  ERR

      CHARACTER BC*1

      CALL FREEC('C',BC,1,1)
      IF((BC.NE.'C').AND.(BC.NE.'G')) THEN
        WRITE(NTW,2000) BC
        WRITE(NOT,2000) BC
        ERR = .TRUE.
        RETURN
      ELSEIF(BC.EQ.'C') THEN
        IA(N+MPKEQ-1) = -IA(N+MPKEQ-1)
      ENDIF
      RETURN

 2000 FORMAT(' **** ERROR: Boundary condition ',A1,' not available.')
      END

C———————————————————————————————————————————————————————FLOELM
      SUBROUTINE FLOELM
C--SUB:FLOELM - COMMAND TO READ & PROCESS FLOW ELEMENT DATA
C
C--HELP LIST——————————————————————————————————————————————————
C
C    .' FLOWELEM                 Flow element command/data group.',/,
C    .' n1 I=n2,n3 E=n4           n1 = element number',/,
C    .'  ...                      n2,n3 = element end nodes',/,
C    .' END                       n4 = filter efficiency',//,
C————————————————————————————————————————————————————————————

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      REAL*8 EFF
      LOGICAL ERR
      EXTERNAL FLOWEO

      ERR = .FALSE.
      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ——— FLOWELEM: FLOW ELEMENTS')
C
C--1.0 CHECK TO SEE IF SYSTEM NODES & EQUATION NUMBERS ARE DEFINED
C
      IF(NFNOD.EQ.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Number of flow system nodes = 0.',/,
     +  '               FLOWSYS command must be executed.')
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 OPEN <filename>.FEL
C
      IF(NFELM.EQ.0)
     +  CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.FEL'),'UNFORMATTED')
      IF(NFELM.GT.0) THEN
        WRITE(NTW,2200)
        WRITE(NOT,2200)
        CALL ABORT
        RETURN
      ENDIF
 2200 FORMAT(' **** ERROR: Flow elements have already been defined.')
C
C--3.0 GET ELEMENT DATA
C
      NELDOF = 2
      CALL ELGEN(FLOWEO,IA(MPKEQ),NELDOF,NFNOD,NFBAN,ERR)
C——— IF ERR ABORT COMMAND
   30 IF(ERR) THEN
        NFELM = 0
        CALL ABORT
        CLOSE(ND1)
        RETURN
      ENDIF
C
C--4.0 REPORT ELEMENT DATA
C
      REWIND(ND1)
      WRITE(NOT,2400)
      IF(ECHO) WRITE(NTW,2400)
 2400 FORMAT(/,'      Elem    I-Node    J-Node    Filter Efficency')
      DO 40 N=1,NFELM
        READ(ND1) LM1,LM2,EFF
        IF(ECHO) WRITE(NTW,2410) N, LM1, LM2, EFF
   40   WRITE(NOT,2410) N, LM1, LM2, EFF
 2410 FORMAT(3(5X,I5),5X, G10.3)
C
C--5.0 CLOSE ELEMENT DATA FILE
C
      CLOSE(ND1)
      RETURN
      END

C———————————————————————————————————————————————————————FLOWEO
      SUBROUTINE FLOWEO(NEL,LM,ERR)
C--SUB:FLOWEO - READS ADDITIONAL ELEMENT DATA
C              WRITES FLOW ELEMENT DATA TO LOGICAL UNIT ND1
```

```
      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      REAL*8 EFF
      INTEGER LM(2),NEL
      LOGICAL ERR
C
C--1.0 GET FILTER EFFICIENCY
C
      EFF = 0.0
      CALL FREER('E',EFF,1)
      IF(EFF.LT.0.0D0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Filter efficiencies must be greater than 0.')
        ERR = .TRUE.
        RETURN
      ENDIF
C
C--2.0 WRITE ELEMENT INFORMATION TO ND1 = <filename.FEL>
C
      WRITE(ND1) LM(1), LM(2), EFF
      NFELM = NEL

      RETURN
      END

C-----------------------------------------------------------------STEADY
      SUBROUTINE STEADY
C--SUB:STEADY - COMMAND TO FORM STEADY PROBLEM [F](C) = (G) & SOLVE
C              SOLUTION (C) IS WRITTEN OVER (G)
C
C--HELP LIST-----------------------------------------------------------
C
C    .' STEADY               Steady state solution.',/,
C    .' n1,n2,n3 W=n4         n1,n2,n3 = elem: first, last, incr.',/,
C    .' ...                   n4 = element flow rate',/,
C    .' n5,n6,n7 CG=n8        n5,n6,n7 = node: first, last, incr.',/,
C    .' ...                   n8 = prescribed conc. or gen. rate',/,
C    .' END',//,
C----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CMDCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      CHARACTER ENDFLAG*3

      ERR = .FALSE.

      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ---- STEADY: STEADY STATE SOLUTION')

C
C--1.0 CHECK IF FLOW SYSTEM AND ELEMENT DATA ARE DEFINED
C
      IF(NFEQN.EQ.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Number of flow system DOFs = 0.',/,
     +  '             FLOWSYS command must be executed.')
        RETURN
      ELSEIF(NFELM.EQ.0) THEN
        WRITE(NTW,2110)
        WRITE(NOT,2110)
 2110   FORMAT(
     +  ' **** ERROR: Number of flow flow elements = 0.',/,
     +  '             FLOWELEM command must be executed.')
        RETURN
      ENDIF
C
C--2.0 DEFINE AND INITIALIZE ARRAYS
C
      CALL DELETE('WE  ')
      CALL DELETE('G   ')
      CALL DELETE('F   ')
      CALL DEFINR('F   ',MPF,NFEQN,2*MFBAN-1)
      CALL DEFINR('G   ',MPG,NFEQN,1)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL ZEROR(IA(MPG),NFEQN,1)
      CALL ZEROR(IA(MPWE),NFELM,1)
C
C--3.0 GET ELEMENT FLOW RATES (WE)
C
      CALL READWE(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--4.0 FORM [F]
C      ALLOW "END" BEFORE EXCITATION DATA TO JUST FORM COMPACT [F]
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND ND1

      CALL FORMF(IA(MPKEQ),IA(MPF),IA(MPWE),'BAND',ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF

      CLOSE(ND1)
      CALL FREEC(' ',ENDFLAG,3,1)
      IF(ENDFLAG.EQ.'END') RETURN
C
C--5.0 FORM (G)
C
      CALL FORMG(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--6.0 MODIFY (G) AND [F] FOR PRESCRIBED CONCENTRATIONS
C
      CALL MODIF(IA(MPKEQ),IA(MPF),IA(MPG),NFNOD,NFEQN,MFBAN)
C
C--7.0 SOLVE
C
      CALL FACTCA(IA(MPF),NFEQN,MFBAN,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
      CALL SOLVCA(IA(MPF),IA(MPG),NFEQN,MFBAN,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--8.0 REPORT SOLUTION
C
      IF(ECHO) WRITE(NTW,2800)
      WRITE(NOT,2800)
 2800 FORMAT(/,'   -- Response: Node Concentrations')
      CALL REPRTN(IA(MPG),IA(MPKEQ),NFEQN,NFNOD)
C
C--9.0 DELETE ARRAYS
C
      CALL DELETE('WE  ')
      CALL DELETE('G   ')
      CALL DELETE('F   ')

      RETURN
      END

C-----------------------------------------------------------------READWE
      SUBROUTINE READWE(ERR)
```

```
C--SUB:READWE - READS & REPORTS ELEMENT TOTAL MASS FLOW RATE DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
      LOGICAL ERR

      EXTERNAL WEDATO

      WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,'   -- Element Mass Flow Rates')
      CALL DATGEN(WEDATO,NFELM,ERR)
      IF(ERR) RETURN

      CALL REPRTE(IA(MPWE),NFELM)

      RETURN
      END

C----------------------------------------------------------------WEDATO
      SUBROUTINE WEDATO(N,ERR)
C--SUB:WEDATO - CALLS WEDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC

      LOGICAL ERR

      CALL WEDAT1(IA(MPWE),NFELM,N)

      RETURN
      END

C----------------------------------------------------------------WEDAT1
      SUBROUTINE WEDAT1(WE,NFELM,N)
C--SUB:WEDATO - READS ELEMENT MASS FLOW RATE DATA
C
      REAL*8 WE(NFELM)

      CALL FREER('W',WE(N),1)

      RETURN
      END

C-----------------------------------------------------------------FORMG
      SUBROUTINE FORMG(ERR)
C--SUB:FORMG - READS & REPORTS NODAL CONTAMINANT GENERATION RATE DATA

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      EXTERNAL GDATO

      WRITE(NOT,2100)
      WRITE(NTW,2100)
 2100 FORMAT(/,
     +'   -- Excitation: Contaminant Concentration or Generation')

      CALL DATGEN(GDATO,NFNOD,ERR)

      CALL REPRTN(IA(MPG),IA(MPKEQ),NFEQN,NFNOD)

      RETURN
      END

C-----------------------------------------------------------------GDATO
      SUBROUTINE GDATO(N,ERR)
C--SUB:GDATO - CALLS GDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC

      LOGICAL ERR

      CALL GDAT1(IA(MPG),IA(MPKEQ),NFEQN,NFNOD,MFBAN,N,ERR)

      RETURN
      END

C-----------------------------------------------------------------GDAT1
      SUBROUTINE GDAT1(G,KEQ,NFEQN,NFNOD,MFBAN,N,ERR)
C--SUB:GDAT1 - READS CONTAMINANT EXCITATION DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC

      REAL*8 G(NFEQN), CDAT, GDAT
      INTEGER KEQ(NFNOD)
      LOGICAL ERR

      CALL FREER('G',GDAT,1)
      NEQ = KEQ(N)
      NNEQ = ABS(NEQ)

      IF(NEQ.NE.0) THEN
        G(NNEQ) = GDAT
      ELSE
        WRITE(NTW,2000) N
        WRITE(NOT,2000) N
 2000   FORMAT(' **** ERROR: Node ',I5,' is not a defined flow node.')
        ERR = .TRUE.
        RETURN
      ENDIF

      RETURN
      END

C-----------------------------------------------------------------MODIF
      SUBROUTINE MODIF(KEQ,F,G,NFNOD,NFEQN,MFBAN)
C--SUB:MODIF - MODIFIES [F] AND {G} FOR C-PRESCRIBED DOFS

      REAL*8 F(NFEQN,2*MFBAN-1),G(NFEQN)
      INTEGER KEQ(NFNOD)

      DO 10 N=1,NFNOD
        NEQ = KEQ(N)
        NNEQ = ABS(NEQ)
        IF(NEQ.LT.0) THEN
           F(NNEQ,MFBAN) = F(NNEQ,MFBAN)*1.0D15
           G(NNEQ) = G(NNEQ)*F(NNEQ,MFBAN)
        ENDIF
   10 CONTINUE
      RETURN
      END

C----------------------------------------------------------------TIMCON
      SUBROUTINE TIMCON
C--SUB:TIMCON - COMMAND TO FORM CONTAM. DISPERSAL EIGENVALUE PROBLEM
C
C               [[V]-1[F] - (1/T)[I]{E} = {0}
C
C               WHERE: [V]   - FLOW VOLUMETRIC MASS MATRIX (DIAGONAL)
C                      [F]   - FLOW SYSTEM FLOW MATRIX
C                      {E}   - (RIGHT) EIGENVECTORS
C                       T    - CONTAM. DISPERSAL TIME CONSTANTS
C
C                 TO EVALUATE TIME CONSTANTS. EIGENVECTORS ARE NOT FOUND.
C--HELP LIST-----------------------------------------------------------
C
C    .' TIMECONS E=n1          Time constant solution, n1 = epsilon',/,
C    .' n2,n3,n4 W=n5           n2,n3,n4 = elem: first, last, incr.',/,
C    .' ...                     n5 = element flow rate',/,
C    .' END')
C----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
      LOGICAL ERR
      CHARACTER ENDFLAG*3

      ERR = .FALSE.
C
C--0.0 WRITE HEADER AND READ PRECISION
C
      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,
     +' ---- TIMECONS: TIME CONSTANTS - CONTAMINANT DISPERSAL SYSTEM ')

      EP1 = EP
      CALL FREER('E',EP1,1)
      WRITE(NOT,2010) EP1
      WRITE(NTW,2010) EP1
 2010 FORMAT(/'      Convergence parameter, epsilon, ...', G10.3)
C
C--1.0 CHECK IF FLOW SYSTEM AND ELEMENT DATA ARE DEFINED
C
       IF(NFEQN.EQ.0) THEN
         WRITE(NTW,2100)
         WRITE(NOT,2100)
 2100    FORMAT(
     +   ' **** ERROR: Number of flow system DOFs = 0.',/,
     +   '             FLOWSYS command must be executed.')
         RETURN
       ELSEIF(NFELM.EQ.0) THEN
         WRITE(NTW,2110)
         WRITE(NOT,2110)
 2110    FORMAT(
     +   ' **** ERROR: Number of flow flow elements = 0.',/,
     +   '             FLOWELEM command must be executed.')
         RETURN
       ENDIF
C
C--2.0 DEFINE AND INITIALIZE ARRAYS
C
       CALL DELETE('WE  ')
       CALL DELETE('V   ')
       CALL DELETE('F   ')
       CALL DEFINR('F   ',MPF,NFEQN,NFEQN)
       CALL DEFINR('V   ',MPV,NFEQN,1)
       CALL DEFINR('WE  ',MPWE,NFELM,1)
       CALL ZEROR(IA(MPV),NFEQN,1)
       CALL ZEROR(IA(MPWE),NFELM,1)
C
C--3.0 GET ELEMENT FLOW RATES (WE)
C
       CALL READWE(ERR)
       IF(ERR) THEN
         CALL ABORT
         RETURN
       ENDIF
C
C--4.0 FORM [F] (ALLOW "END" BEFORE VOL. MASS DATA TO JUST FORM [F])
C
       OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
      +FORM='UNFORMATTED')
       REWIND ND1

       CALL FORMF(IA(MPKEQ),IA(MPF),IA(MPWE),'FULL',ERR)
       IF(ERR) THEN
         CALL ABORT
         RETURN
       ENDIF

       CLOSE(ND1)
       CALL FREEC(' ',ENDFLAG,3,1)
       IF(ENDFLAG.EQ.'END') RETURN
C
C--5.0 GET NODAL VOLUMETRIC MASS AND REPORT NOMINAL TIME CONSTANTS
C
       CALL READV(ERR)
       IF(ERR) THEN
         CALL ABORT
         RETURN
       ENDIF

       IF(ECHO) WRITE(NTW,2500)
      WRITE(NOT,2500)
 2500 FORMAT(/,'   -- Nominal Time Constants')

      CALL REPRTT(IA(MPF),IA(MPV),NFEQN,1)
C
C--6.0 PREMULTIPLY [F] BY [V] INVERSE
C
      CALL VINVF(IA(MPF),IA(MPV),NFEQN,EP,ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--7.0 SOLVE EIGENVALUE PROBLEM
C
      IF(ECHO) WRITE(NTW,2700)
      WRITE(NOT,2700)
 2700 FORMAT(/,'   -- Actual Time Constants')
      WRITE(NTW,2710)
 2710 FORMAT(/,'   -- NOTE: Computation of actual time constants ',
     +'may take considerable time.')
      NIT = 0

      CALL EIGEN2(IA(MPF),IA(MPF),NFEQN,NIT,EP1)
C
C--8.0 REPORT TIME CONSTANTS & ITERATION INFORMATION
C
      CALL REPRTT(IA(MPF),IA(MPV),NFEQN,2)
      WRITE(NTW,2800) ABS(NIT)
      WRITE(NOT,2800) ABS(NIT)
 2800 FORMAT(/'      Number of iterations used ...',I5)
      IF((NIT.LT.0).OR.(NAIT.EQ.50)) THEN
        WRITE(NTW,2810)
        WRITE(NOT,2810)
 2810 FORMAT(' **** WARNING: Procedure did not converge.')
      ENDIF
C
C--9.0 DELETE ARRAYS
C
      CALL DELETE('WE  ')
      CALL DELETE('V   ')
      CALL DELETE('F   ')

      RETURN
      END

C--------------------------------------------------------------VINVF
      SUBROUTINE VINVF(F,V,NFEQN,EP,ERR)
C--SUB: VINVF: EVALUATES [V]-1[F] : CALLED BY TIMCON

      INCLUDE IOCOM.INC

      REAL*8 F(NFEQN,1), V(NFEQN), EP, EPZERO
      LOGICAL ERR
C
C--1.0 FIND MAX VOLUMETRIC MASS TO ESTABLISH RELATIVE MACHINE ZERO
C
      VMAX = 0.0D0
      DO 10 I=1,NFEQN
       IF(V(I).GT.VMAX) VMAX=V(I)
   10 CONTINUE
      EPZERO = EP*VMAX
C
C--2.0 EVALUATE PRODUCT [V]-1[F]: ERR IF DIV BY MACHINE ZERO
C
      DO 20 I=1,NFEQN
       VII = V(I)
       IF(VII.LE.EPZERO) THEN
         WRITE(NTW,2000) I
         WRITE(NOT,2000) I
         ERR = .TRUE.
         RETURN
       ENDIF
 2000    FORMAT(
     +' **** ERROR: Volumetric mass less than relative machine zero.',/,
     +'             Equation number: ',I5)
       DO 20 J=1,NFEQN
        F(I,J) = F(I,J)/VII
   20 CONTINUE
```

```
      RETURN
      END

C-----------------------------------------------------------------REPRTT
      SUBROUTINE REPRTT(F,V,NFEQN,OPT)
C--SUB:REPRTT - REPORTS TIME CONSTANTS: CALLED BY TIMCON

      INCLUDE IOCOM.INC

      REAL*8 F(NFEQN,1),V(NFEQN)
      INTEGER OPT

      IF(OPT.EQ.1) THEN
C
C--1.0 REPORT NOMINAL TIME CONSTANTS V(I,I)/F(I,I)
C
      WRITE(NOT,2010)
      IF(ECHO) WRITE(NTW,2010)
      WRITE(NOT,2020) (N, V(N)/F(N,N), N=1,NFEQN)
      IF(ECHO) WRITE(NTW,2020) (N, V(N)/F(N,N), N=1,NFEQN)

      ELSE
C
C--2.0 REPORT ACTUAL TIME CONSTANTS
C
      WRITE(NOT,2040)
      IF(ECHO) WRITE(NTW,2040)
      WRITE(NOT,2020) (N, 1.0D0/F(N,N), N=1,NFEQN)
      IF(ECHO) WRITE(NTW,2020) (N, 1.0D0/F(N,N), N=1,NFEQN)

      ENDIF

 2010 FORMAT(/,6X,4(2X,'Node    Value',3X))
 2020 FORMAT((6X,4(I6,1X,G11.3)))
 2040 FORMAT(/,6X,4(2X,'Num.    Value',3X))

      RETURN
      END

C-----------------------------------------------------------------FLODAT
      SUBROUTINE FLODAT
C--SUB:FLODAT - COMMAND TO READ ELEMENT FLOW DATA & GENERATE STEPWISE
C               TIME HISTORIES OF FLOW DATA AND WRITES TIME HISTORIES
C               IN FORMAT;
C
C                 TIME
C                 (WE(I),I=1,NFELM)
C                 TIME
C                 (WE(I),I=1,NFELM)
C                     ...
C
C                TO FILE <filename>.WDT
C
C               OPTIONALLY EQUAL STEP TIME HISTORIES MAY BE GENERATED
C--HELP LIST-----------------------------------------------------------
C
C    .' FLOWDAT [T=n1,n2,n3]  Generate element flow time histories.',/,
C    .' TIME=n1                 n1 = time',/,
C    .' n1,n2,n3 W=n4           n1,n2,n3 = node: first, last, incr.',/,
C    .' ...                     n4 = element mass flow rate.',/,
C    .' :',/,
C    .' END',//,
C
C----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)
C
C-- CAL-SAP: DATA & COMMON STORAGE
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

C
C-- FLODAT: DATA & COMMON STORAGE
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S -----------------
C
C  POINTER   VARIABLE           DESCRIPTION
C
C              TIME(3)              : START TIME, ENDTIME, TIMESTEP
C  MPWE      WE(NFELM)            : CURRENT ELEMENT MASS FLOW VALUES
C
C     T I M E   H I S T O R Y   D A T A
C
C     DAT(1) |          * - - -  Time histories of excitation data are
C            |          |        defined as step-wise functions of time
C            |        *-         using arbitrary values or, optionally,
C            |        |          generated intermediate values of
C            |      *-           equal step size.
C            |      |
C     DAT(2) |- - *-
C            |----|-----|--------
C                 TM(2) TM(1)
C
C   MPTDAT   TDAT(2)             : CURRENT ARBITRARY TIME VALUES
C   MPWDAT   WDAT(NFELM,2)       : CORRESPONDING ELEM. FLOW DATA
C----------------------------------------------------------------------
      COMMON /FLOOT/ MPTDAT,MPWDAT
      REAL*8 TIME(3)
      LOGICAL ERR
      CHARACTER ENDFLAG*3

      ERR = .FALSE.
      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ---- FLOWDAT: ELEMENT FLOW TIME HISTORY DATA')
C
C--1.0 CHECK TO SEE IF ELEMENTS HAVE BEEN DEFINED
C
      IF(NFELM.EQ.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Number of flow elements = 0.',/,
     +  '             FLOWELEM command must be executed.')
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 GET DATA GENERATION CONTROL DATA
C
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      IF(TIME(3).LT.0.0D0) THEN
        WRITE(NTW,2200)
        WRITE(NOT,2200)
 2200   FORMAT(' **** ERROR: Time step may not be negative.')
        CALL ABORT
        RETURN
      ELSEIF(TIME(3).GT.0.0D0) THEN
        IF(TIME(2).LT.TIME(1)) THEN
          WRITE(NTW,2210)
          WRITE(NOT,2210)
 2210     FORMAT(
     +' **** ERROR: Final time must be greater than initial time.')
          CALL ABORT
          RETURN
        ENDIF

        IF(ECHO) WRITE(NTW,2220)
        WRITE(NOT,2220)
 2220   FORMAT(/,'  -- Generation Control Variables')
        IF(ECHO) WRITE(NTW,2230) (TIME(I),I=1,3)
        WRITE(NOT,2230) (TIME(I),I=1,3)
 2230   FORMAT(/,
     .'      Initial time ..................... ',G10.3,/,
     .'      Final time ....................... ',G10.3,/,
     .'      Time step increment ............. ',G10.3)
      ENDIF
C
C--3.0 OPEN <filename>.WDT
C
      CALL WOPEN(WD1,(FNAME(1:LFNAME)//'.WDT'),'UNFORMATTED')
C
C--4.0 READ & GENERATE FLOW DATA
C
```

```
      WRITE(NOT,2400)
      WRITE(NTW,2400)
 2400 FORMAT(/,'   -- Element Mass Flow Time History Data')
C
C---4.1 DEFINE & INITIALIZE ARRAYS
C
      CALL DELETE('TDAT')
      CALL DELETE('WDAT')
      NWDAT = 1
      IF(TIME(3).GT.0.0D0) THEN
        CALL DELETE('WE  ')
        CALL DEFINR('WE  ',MPWE,NFELM,1)
        CALL ZEROR(IA(MPWE),NFELM,1)
        NWDAT = 2
      ENDIF
      CALL DEFINR('WDAT',MPWDAT,NFELM,NWDAT)
      CALL DEFINR('TDAT',MPTDAT,1,2)
      CALL ZEROR(IA(MPWDAT),NFELM,NWDAT)
      CALL ZEROR(IA(MPTDAT),1,2)
C
C---4.2 GENERATE VALUES & WRITE TO <filename>.WDT
C
      IF(TIME(3).GT.0.0D0) THEN
        CALL GENWD1(IA(MPWE),IA(MPTDAT),IA(MPWDAT),TIME,ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ELSE
        CALL GENWD2(IA(MPTDAT),IA(MPWDAT),ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ENDIF

C
C--5.0 DELETE ARRAYS, CLOSE ELEMENT FLOW DATA FILE, SKIP TO "END"
C
      CALL DELETE('TDAT')
      CALL DELETE('WDAT')
      CALL DELETE('WE  ')

      CLOSE(WD1)

      IF(MODE.EQ.'BATCH') THEN
  500   IF(EOC) RETURN
        CALL FREE
        GO TO 500
      ENDIF

      RETURN
      END

C----------------------------------------------------------------GENWD1
      SUBROUTINE GENWD1(WE,TDAT,WDAT,TIME,ERR)
C--SUB: GENWD1 - GENERATES ELEMENT MASS FLOW DATA, AT EQUAL TIME STEP
C               INTERVALS, FROM GIVEN ARBITRARY DISCRETE TIME DATA
C----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
C
C---- FLOWDAT: DATA & COMMON STORAGE
C
      COMMON /FLOUT/ MPTDAT,MPWDAT
      LOGICAL ERR
C
C---- GENWD1: DATA & COMMON STORAGE
C
      REAL*8 WE(NFELM),TDAT(2),WDAT(NFELM,2),TIME(3)
C
C--1.0 GET FIRST TWO TIME HISTORY RECORDS ( TDAT(1), WDAT(NFELM,1) )
C
      CALL GETTDT(TDAT)
      IF(EOC) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(' **** ERROR: Insufficient data.')
        ERR = .TRUE.
        RETURN
      ENDIF
      CALL GETWDT(WDAT,ERR)
      IF(ERR) RETURN

      CALL GETTDT(TDAT)
      IF(EOC) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
        ERR = .TRUE.
        RETURN
      ELSEIF(TDAT(1).LT.TDAT(2)) THEN
        WRITE(NTW,2110)
        WRITE(NOT,2110)
 2110   FORMAT(' **** ERROR: Time data out of sequence.')
        ERR = .TRUE.
        RETURN
      ENDIF
      CALL GETWDT(WDAT,ERR)
      IF(ERR) RETURN
C
C--2.0 GENERATION TIME LOOP
C
      DO 200 T=TIME(1),TIME(2),TIME(3)
C
C---2.1 UPDATE EXCITATION FUNCTION DATA IF NEEDED
C
   20 IF(T.GT.TDAT(1)) THEN
        CALL GETTDT(TDAT)
        IF(EOC) THEN
          WRITE(NTW,2100)
          WRITE(NOT,2100)
          ERR = .TRUE.
          RETURN
        ELSEIF(TDAT(1).LT.TDAT(2)) THEN
          WRITE(NTW,2110)
          WRITE(NOT,2110)
          ERR = .TRUE.
          RETURN
        ENDIF
        CALL GETWDT(WDAT,ERR)
        IF(ERR) RETURN
        GO TO 20
      ENDIF
C
C---2.2 COMPUTE INTERPOLATION FRACTION
C
      XT = (T-TDAT(2))/(TDAT(1)-TDAT(2))
C
C---2.3 COMPUTE {WE(T)}
C
      CALL ZEROR(WE,NFELM,1)

      DO 23 N=1,NFELM
      WE(N) = WDAT(N,2) + XT*(WDAT(N,1)-WDAT(N,2))
   23 CONTINUE
C
C---2.4 WRITE TIME,{WE(T)} TO WD1
C

      WRITE(WD1) T
      WRITE(WD1) (WE(I),I=1,NFELM)

  200 CONTINUE
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
      WRITE(WD1) T

      RETURN
      END

C----------------------------------------------------------------GETTDT
      SUBROUTINE GETTDT(TDAT)
C--SUB: GETTDO - UPDATES TIME DATA VALUES
C
      INCLUDE IOCOM.INC
      REAL*8 TDAT(2)
C
C--1.0 UPDATE OLD VALUES
C
      TDAT(2) = TDAT(1)
C
C--2.0 READ NEW VALUE
C
C---- CHECK FOR END-OF-COMMAND "END"
      IF(EOC) THEN
        EOD = .TRUE.
        RETURN
      ENDIF
      IF(MODE.EQ.'INTER') CALL PROMPT(' TIME>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREENR(NTW)
C---- CHECK FOR END-OF-COMMAND "END"
      IF(EOC) THEN
        EOD = .TRUE.
        RETURN
      ENDIF
      CALL FREER('E',TDAT(1),1)
C---- REPORT
      IF(ECHO) WRITE(NTW,2020) TDAT(1)
      WRITE(NOT,2020) TDAT(1)
 2020 FORMAT(/,'    -- Time: ',G10.3)

      RETURN
      END

C--------------------------------------------------------------GETWDT
      SUBROUTINE GETWDT(WDAT,ERR)
C--SUB: GETWDT - UPDATES ELEMENT FLOW DATA VALUES
C
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      REAL*8 WDAT(NFELM,2)
      EXTERNAL WDAT0
C
C--1.0 UPDATE 'OLD' DATA VALUES; INITIALIZE 'NEW' DATA VALUES
C
      DO 10 N=1,NFELM
      WDAT(N,2) = WDAT(N,1)
   10 WDAT(N,1) = 0.0D0
C
C--2.0 READ NEW VALUES
C
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN

      CALL REPRTE(WDAT(1,1),NFELM)

      RETURN
      END

C---------------------------------------------------------------WDAT0
      SUBROUTINE WDAT0(N,ERR)
C--SUB:WDAT0 - CALLS WDAT11 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC

      COMMON /FLODT/ MPTDAT,MPWDAT
      LOGICAL ERR

      CALL WDAT1(IA(MPWDAT),NFELM,N)

      RETURN
      END

C---------------------------------------------------------------WDAT1
      SUBROUTINE WDAT1(WDAT,NFELM,N)
C--SUB:WDAT1 - READS ELEMENT MASS FLOW RATE TIME HISTORY DATA
C
      REAL*8 WDAT(NFELM,1)

      CALL FREER('W',WDAT(N,1),1)
```

```
      RETURN
      END

C--------------------------------------------------------------GENWD2
      SUBROUTINE GENWD2(TDAT,WDAT,ERR)
C--SUB: GENWD2 - GENERATES ELEMENT MASS FLOW DATA, AT GIVEN TIME STEP
C                INTERVALS, FROM GIVEN DISCRETE TIME DATA
C---------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
C
C---- FLOWDAT: DATA & COMMON STORAGE
C
      COMMON /FLODT/ MPTDAT,MPWDAT
      LOGICAL ERR
      EXTERNAL WDAT0
C
C---- GENWD2: DATA & COMMON STORAGE
C
      REAL*8 TDAT(2),WDAT(NFELM,1)

C
C--1.0 GET FIRST TIME HISTORY RECORD ( TDAT(1), WDAT(NFELM,1) )
C
      CALL GETTDT(TDAT)
      IF(EOC) RETURN
      TDAT(2) = TDAT(1)
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN
      CALL REPRTE(WDAT(1,1),NFELM)
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (WDAT(I,1),I=1,NFELM)
C
C--2.0 GET ADDITIONAL TIME HISTORY RECORDS
C
   20 CALL GETTDT(TDAT)
      IF(EOC) GO TO 300
      IF(TDAT(1).LT.TDAT(2)) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(' **** ERROR: Time data out of sequence.')
        ERR = .TRUE.
        RETURN
      ENDIF
      TDAT(2) = TDAT(1)
      CALL DATGEN(WDAT0,NFELM,ERR)
      IF(ERR) RETURN
      CALL REPRTE(WDAT(1,1),NFELM)
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (WDAT(I,1),I=1,NFELM)
      GO TO 20
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
  300 WRITE(ND1) TDAT(1)

      RETURN
      END

C--------------------------------------------------------------EXCDAT
      SUBROUTINE EXCDAT
C--SUB:EXCDAT - COMMAND TO READ EXCITATION DATA & GENERATE STEPWISE
C               TIME HISTORIES OF EXCITATION VALUES, E(NFEQN),AND
C               WRITES TIME HISTORIES IN FORMAT;
C
C                  TIME
C                  (E(I),I=1,NFEQN)
C                  TIME
C                  (E(I),I=1,NFEQN)
C                      ...
C
C               TO FILE <filename>.EDT
C--HELP LIST----------------------------------------------------------
C
C     .' EXCITDAT [T=n1,n2,n3] Generate excitation time histories.',/,
C     .' TIME=n1               n1 = time',/,
C     .' n1,n2,n3 CG=n4        n1,n2,n3 = node: first, last, incr.',/,
C    .' ....                        n4 = excitation: conc. or gen. rate.',/,
C    .' :',/,
C    .' END',//,
C
C------------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,MP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

C-- EXCDAT: DATA & COMMON STORAGE ----------------------------------------
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S --------------------
C
C  POINTER   VARIABLE           DESCRIPTION
C
C            TIME(3)            : START TIME, ENDTIME, TIMESTEP
C  MPE       E(NFELM)           : CURRENT EXCITATION VALUES
C
C    T I M E   H I S T O R Y   D A T A
C
C    DAT(1) |           * - - -  Time histories of excitation data are
C           |           |        defined as step-wise functions of time
C           |        *-          using arbitrary values or, optionally,
C           |        |           generated intermediate values of
C           |     *-             equal step size.
C           |     |
C    DAT(2) |- - *-
C           |----|-----|------..
C                TM(2) TM(1)
C
C  MPTDAT   TDAT(2)             : CURRENT ARBITRARY TIME VALUES
C  MPEDAT   EDAT(NFNOD,2)       : CORRESPONDING EXCITATION DATA
C------------------------------------------------------------------------
      COMMON /EXCDT/ MPTDAT,MPEDAT
      REAL*8 TIME(3)
      CHARACTER ENDFLAG*3
      LOGICAL ERR

      ERR = .FALSE.
      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ---- EXCITDAT: EXCITATION TIME HISTORY DATA')

C
C--1.0 CHECK TO SEE IF FLOW SYSTEM HAS BEEN DEFINED
C
      IF(MFEQN.EQ.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Number of flow system DOFs = 0.',/,
     +  '                FLOWSYS command must be executed.')
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 GET DATA GENERATION CONTROL DATA
C
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      IF(TIME(3).LT.0.0D0) THEN
        WRITE(NTW,2200)
        WRITE(NOT,2200)
 2200   FORMAT(' **** ERROR: Time step may not be negative.')
        CALL ABORT
        RETURN
      ELSEIF(TIME(3).GT.0.0D0) THEN
        IF(TIME(2).LT.TIME(1)) THEN
          WRITE(NTW,2210)
          WRITE(NOT,2210)
 2210     FORMAT(
     +' **** ERROR: Final time must be greater than initial time.')
          CALL ABORT
          RETURN
        ENDIF

        IF(ECHO) WRITE(NTW,2220)
        WRITE(NOT,2220)
 2220   FORMAT(/,'  -- Generation Control Variables')
        IF(ECHO) WRITE(NTW,2230) (TIME(I),I=1,3)
        WRITE(NOT,2230) (TIME(I),I=1,3)
 2230   FORMAT(/,
     .'      Initial time .................... ',G10.3,/,
     .'      Final time ...................... ',G10.3,/,
     .'      Time step increment ............. ',G10.3)
      ENDIF
C
C--3.0 OPEN <filename>.EDT
C
      CALL NOPEN(ND1,(FNAME(1:LFNAME)//'.EDT'),'UNFORMATTED')
C
C--4.0 READ & GENERATE EXCITATION DATA
C
      WRITE(NOT,2400)
      IF(ECHO) WRITE(NTW,2400)
 2400 FORMAT(/,'   -- Nodal Excitation Time History Data')
C
C---4.1 DEFINE & INITIALIZE ARRAYS
C
      CALL DELETE('TDAT')
      CALL DELETE('EDAT')
      CALL DELETE('E   ')
      CALL DEFINR('E   ',MPE,MFEQN,1)
      CALL ZEROR(IA(MPE),MFEQN,1)
      NEDAT = 1
      IF(TIME(3).GT.0.0D0)  NEDAT = 2
      CALL DEFINR('EDAT',MPEDAT,NFNOD,NEDAT)
      CALL DEFINR('TDAT',MPTDAT,1,2)
      CALL ZEROR(IA(MPEDAT),NFNOD,NEDAT)
      CALL ZEROR(IA(MPTDAT),1,2)
C
C---4.2 GENERATE VALUES & WRITE TO <filename>.EDT
C
      IF(TIME(3).GT.0.0D0) THEN
        CALL GENED1(IA(MPKEQ),IA(MPE),IA(MPTDAT),IA(MPEDAT),TIME,ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ELSE
        CALL GENED2(IA(MPKEQ),IA(MPE),IA(MPTDAT),IA(MPEDAT),ERR)
        IF(ERR) THEN
          CALL ABORT
          RETURN
        ENDIF
      ENDIF
C
C--5.0 DELETE ARRAYS, CLOSE ELEMENT FLOW DATA FILE, SKIP TO "END"
C
      CALL DELETE('TDAT')
      CALL DELETE('EDAT')
      CALL DELETE('E   ')

      CLOSE(ND1)

      IF(MODE.EQ.'BATCH') THEN
  500   IF(EOC) RETURN
        CALL FREE
        GO TO 500
      ENDIF

      RETURN
      END

C-----------------------------------------------------------------GENED1
      SUBROUTINE GENED1(KEQ,E,TDAT,EDAT,TIME,ERR)
C--SUB: GENED1 - GENERATES EXCITATION DATA, AT EQUAL TIME STEP
C                INTERVALS, FROM GIVEN ARBITRARY TIME DATA
C------------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
      LOGICAL ERR
C
C---- GENED1: DATA & COMMON STORAGE
C
      REAL*8 E(NFEQN),TDAT(2),EDAT(NFMOD,2),TIME(3)
      INTEGER KEQ(NFMOD)
C
C--1.0 GET FIRST TWO TIME HISTORY RECORDS ( TDAT(2), EDAT(NFMOD,2) )
C
       CALL GETTDT(TDAT)
       IF(EOC) THEN
         WRITE(NTW,2100)
         WRITE(NOT,2100)
 2100    FORMAT(' **** ERROR: Insufficient data.')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(EDAT,ERR)
       IF(ERR) RETURN

       CALL GETTDT(TDAT)
       IF(EOC) THEN
         WRITE(NTW,2100)
         WRITE(NOT,2100)
         ERR = .TRUE.
         RETURN
       ELSEIF(TDAT(1).LT.TDAT(2)) THEN
         WRITE(NTW,2110)
         WRITE(NOT,2110)
 2110    FORMAT(' **** ERROR: Time data out of sequence.')
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(EDAT,ERR)
       IF(ERR) RETURN
C
C--2.0 GENERATION TIME LOOP
C
      DO 200 T=TIME(1),TIME(2),TIME(3)
C
C---2.1 UPDATE EXCITATION FUNCTION DATA IF NEEDED
C
   20 IF(T.GT.TDAT(1)) THEN
       CALL GETTDT(TDAT)
       IF(EOC) THEN
         WRITE(NTW,2100)
         WRITE(NOT,2100)
         ERR = .TRUE.
         RETURN
       ELSEIF(TDAT(1).LT.TDAT(2)) THEN
         WRITE(NTW,2110)
         WRITE(NOT,2110)
         ERR = .TRUE.
         RETURN
       ENDIF
       CALL GETEDT(EDAT,ERR)
       IF(ERR) RETURN
       GO TO 20
      ENDIF
C
C---2.2 COMPUTE INTERPOLATION FRACTION
C
      XT = (T-TDAT(2))/(TDAT(1)-TDAT(2))
C
C---2.3 COMPUTE {E(T)}
C
      CALL ZEROR(E,NFMOD,1)

      DO 23 N=1,NFMOD
        NEQ = ABS(KEQ(N))
        IF(NEQ.NE.0) E(NEQ) = EDAT(N,2) + XT*(EDAT(N,1)-EDAT(N,2))
   23 CONTINUE
C
C---2.4 WRITE TIME,{E(T)} TO ND1
C

      WRITE(ND1) T
      WRITE(ND1) (E(I),I=1,NFEQN)
```

```
  200 CONTINUE
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
      WRITE(ND1) T

      RETURN
      END

C------------------------------------------------------------------GETEDT
      SUBROUTINE GETEDT(EDAT,ERR)
C--SUB: GETEDT - UPDATES EXCITATION DATA VALUES
C------------------------------------------------------------------------

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC
C
C-- GETEDT: DATA & COMMON STORAGE
C
      LOGICAL ERR
      REAL*8 EDAT(NFMOD,2)
      EXTERNAL EDATO
C
C--1.0 UPDATE 'OLD' DATA VALUES; INITIALIZE 'NEW' DATA VALUES
C
      DO 10 N=1,NFMOD
      EDAT(N,2) = EDAT(N,1)
   10 EDAT(N,1) = 0.0D0
C
C--2.0 READ NEW VALUES
C
      CALL DATGEN(EDATO,NFMOD,ERR)
      IF(ERR) RETURN

      CALL REPRTN(EDAT(1,1),IA(MPKEQ),NFMOD,NFMOD)

      RETURN
      END

C-------------------------------------------------------------------EDATO
      SUBROUTINE EDATO(N,ERR)
C--SUB:EDATO - CALLS EDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)
      INCLUDE CNTCOM86.INC
      COMMON /EXCDT/ MPTDAT,MPEDAT
      LOGICAL ERR

      CALL EDAT1(IA(MPEDAT),NFMOD,N)

      RETURN
      END

C-------------------------------------------------------------------EDAT1
      SUBROUTINE EDAT1(EDAT,NFMOD,N)
C--SUB:EDATO - READS EXCITATION TIME HISTORY DATA
C
      REAL*8 EDAT(NFMOD,1)

      CALL FREER('G',EDAT(N,1),1)

      RETURN
      END

C------------------------------------------------------------------GENED2
      SUBROUTINE GENED2(KEQ,E,TDAT,EDAT,ERR)
C--SUB: GENED2 - GENERATES EXCITATION DATA FROM GIVEN TIME DATA
C------------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      EXTERNAL EDATO
```

```
C
C---- GENED2: DATA & COMMON STORAGE
C
      REAL*8 TDAT(2),EDAT(NFNOD,1), E(NFEQN)
      INTEGER KEQ(NFNOD)
C
C--1.0 GET FIRST TIME HISTORY RECORD ( TDAT(1), EDAT(NFNOD,1) )
C
      CALL GETTDT(TDAT)
      IF(EOC) RETURN
      TDAT(2) = TDAT(1)
      CALL DATGEN(EDATO,NFNOD,ERR)
      IF(ERR) RETURN
      DO 10 N=1,NFNOD
        NEQ = ABS(KEQ(N))
   10 IF(NEQ.NE.0) E(NEQ) = EDAT(N,1)
      CALL REPRTN(E,IA(MPKEQ),NFEQN,NFNOD)
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (E(I),I=1,NFEQN)
C
C--2.0 GET ADDITIONAL TIME HISTORY RECORDS
C
   20 CALL GETTDT(TDAT)
      IF(EOC) GO TO 300
      IF(TDAT(1).LT.TDAT(2)) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(' **** ERROR: Time data out of sequence.')
        ERR = .TRUE.
        RETURN
      ENDIF
      TDAT(2) = TDAT(1)
      CALL DATGEN(EDATO,NFNOD,ERR)
      DO 22 N=1,NFNOD
        NEQ = ABS(KEQ(N))
   22 IF(NEQ.NE.0) E(NEQ) = EDAT(N,1)
      CALL REPRTN(E,IA(MPKEQ),NFEQN,NFNOD)
      WRITE(ND1) TDAT(1)
      WRITE(ND1) (E(I),I=1,NFEQN)
      GO TO 20
C
C--3.0 WRITE ONE ADDITIONAL TIME VALUE TO DISK
C
 300  WRITE(ND1) TDAT(1)

      RETURN
      END

C-----------------------------------------------------------------DYNAM
      SUBROUTINE DYNAM
C--SUB:DYNAM - COMMAND TO FORM & SOLVE DYNAMIC PROBLEM
C
C               [F(t)]{C} + [V]d{C}/dt = {G(t)}
C
C            * EXCITATION, {G} AND PRESCRIBED {C}, UPDATED AT DISCRETE
C              TIMES USED TO DEFINE EXCITATION (READ FROM ND1)
C            * FLOW MATRIX, [F], UPDATED AT DISCRETE TIMES USED TO
C              DEFINED ELEMENT FLOW RATES (READ FROM ND2)
C--HELP LIST-----------------------------------------------------------
C    .' DYNAMIC                Dynamic solution.',/,
C    .' T=n1,n2,n3 A=n4         n1,n2,n3 = init,final,incr; n4 =alpha',/,
C    .' n5,n6,n7 IC=n8          n5,n6,n7 = node: first, last, incr.',/,
C    .' ...                     n8 = nodal initial concentrations',/,
C    .' :',/,
C    .' END',//,
C----------------------------------------------------------------------

      IMPLICIT REAL*8(A-H,O-Z)

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      COMMON /DYNM/ TWDAT,TEDAT
      LOGICAL ERR, FOUND
      REAL*8 TIME(3), PSCALE
      INTEGER PINT
C
```

```
C---- DICTIONARY OF VARIABLES ---------------------
C
C     VARIABLE       DESCRIPTION---------------------------------------
C     TIME(3)        START TIME, END TIME, TIME INCREMENT
C     TWDAT          TIME OF NEXT ELEMENT FLOW RATE RECORD
C     TEDAT          TIME OF NEXT EXCITATION RECORD
C     PINT           RESPONSE RESULTS PRINT INTERVAL
C     PSCALE         RESULTS PLOT FILE SCALE FACTOR
C
C     POINTERS TO BLANK COMMON LOCATIONS
C
C     MPFS   FS(NFEQN,2*NFBAN-1): [F*] DYNAM ALG. MATRIX (ASYM-COMPACT)
C     MPC            C(NFEQN)     : CURRENT {C}
C     MPCD          CD(NFEQN)     : CURRENT d{C}/dt
C     MPCDD        CDD(NFEQN)     : CURRENT d/dt(d{C}/dt)
C     MPG            G(NFEQN)     : CURRENT {G}
C----------------------------------------------------------------------

      ERR = .FALSE.

      WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(/,' ---- DYNAMIC: DYNAMIC SOLUTION')

C
C--1.0 CHECK IF SYSTEM, ELEMENT, AND EXCITATION DATA ARE DEFINED & AVAIL
C
      IF(NFEQN.EQ.0) THEN
        WRITE(NTW,2100)
        WRITE(NOT,2100)
 2100   FORMAT(
     +  ' **** ERROR: Number of flow system DOFs = 0.',/,
     +  '             FLOWSYS command must be executed.')
        CALL ABORT
        RETURN
      ELSEIF(NFELM.EQ.0) THEN
        WRITE(NTW,2110)
        WRITE(NOT,2110)
 2110   FORMAT(
     +  ' **** ERROR: Number of flow elements = 0.',/,
     +  '             FLOWELEM command must be executed.')
        CALL ABORT
        RETURN
      ENDIF

      INQUIRE(FILE=(FNAME(1:LFNAME)//'.FEL'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        WRITE(NTW,2120) (FNAME(1:LFNAME)//'.FEL')
        WRITE(NOT,2120) (FNAME(1:LFNAME)//'.FEL')
 2120   FORMAT(' **** ERROR: Element data file ',A,' not found.',/,
     +         '             FLOWELEM command must be executed.')
        CALL ABORT
        RETURN
      ENDIF

      INQUIRE(FILE=(FNAME(1:LFNAME)//'.WDT'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        WRITE(NTW,2130) (FNAME(1:LFNAME)//'.WDT')
        WRITE(NOT,2130) (FNAME(1:LFNAME)//'.WDT')
 2130   FORMAT(' **** ERROR: Flow data file ',A,' not found.',/,
     +         '             FLOWDAT command must be executed.')
        CALL ABORT
        RETURN
      ENDIF

      INQUIRE(FILE=(FNAME(1:LFNAME)//'.EDT'),EXIST=FOUND)
      IF(.NOT.FOUND) THEN
        WRITE(NTW,2140) (FNAME(1:LFNAME)//'.EDT')
        WRITE(NOT,2140) (FNAME(1:LFNAME)//'.EDT')
 2140   FORMAT(' **** ERROR: Excitation data file ',A,' not found.',/,
     +         '             EXCITDAT command must be executed.')
        CALL ABORT
        RETURN
      ENDIF
C
C--2.0 GET DYNAMIC SOLUTION CONTROL VARIABLES
C
      WRITE(NTW,2200)
      WRITE(NOT,2200)
 2200 FORMAT(/'   -- Solution Control Variables')

      IF(MODE.EQ.'INTER') CALL PROMPT(' DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREENR(NTW)
      TIME(1) = 0.0D0
      TIME(2) = 0.0D0
      TIME(3) = 0.0D0
      CALL FREER('T',TIME(1),3)
      IF(TIME(3).LE.0.0D0) THEN
        WRITE(NTW,2210)
        WRITE(NOT,2210)
 2210   FORMAT(' **** ERROR: Time step must be greater than 0.0.')
        CALL ABORT
        RETURN
      ELSEIF(TIME(2).LT.TIME(1)) THEN
        WRITE(NTW,2220)
        WRITE(NOT,2220)
 2220 FORMAT(
     +' **** ERROR: Final time must be greater than initial time.')
        CALL ABORT
        RETURN
      ENDIF

      ALPHA = 0.75D0
      CALL FREER('A',ALPHA,1)
      IF((ALPHA.LT.0.0D0).OR.(ALPHA.GT.1.0D0)) THEN
        WRITE(NTW,2230)
        WRITE(NOT,2230)
 2230   FORMAT(' **** ERROR: Alpha must be in range 0.0 to 1.0.')
        CALL ABORT
        RETURN
      ENDIF

      PINT = 1
      CALL FREEI('I',PINT,1)
      IF(PINT.LT.0) THEN
        WRITE(NTW,2240)
        WRITE(NOT,2240)
 2240   FORMAT(' **** ERROR: Results print interval must be > 0.')
        CALL ABORT
        RETURN
      ENDIF

      PSCALE = 0.0D0
      CALL FREER('S',PSCALE,1)

      IF(ECHO) WRITE(NTW,2250) (TIME(I),I=1,3),ALPHA,PINT
      WRITE(NOT,2250) (TIME(I),I=1,3),ALPHA,PINT
 2250 FORMAT(/,
     .'      Initial time ..................... ',G10.3,/,
     .'      Final time ....................... ',G10.3,/,
     .'      Time step increment .............. ',G10.3,/,
     .'      Integration parameter: alpha .... ',G10.3,/,
     .'      Results print interval .......... ',I6)
      IF(PSCALE.NE.0.0D0) THEN
        IF(ECHO) WRITE(NTW,2260) PSCALE
        WRITE(NOT,2260) PSCALE
      ENDIF
 2260 FORMAT('      Results plot-file scale factor .. ',G10.3)

C
C--3.0 DEFINE AND INITIALIZE SYSTEM ARRAYS
C
      CALL DELETE('WE  ')
      CALL DELETE('C   ')
      CALL DELETE('G   ')
      CALL DELETE('F   ')
      CALL DELETE('V   ')
      CALL DEFINR('V   ',MPV,NFEQN,1)
      CALL DEFINR('F   ',MPF,NFEQN,2*NFBAN-1)
      CALL DEFINR('G   ',MPG,NFEQN,1)
      CALL DEFINR('C   ',MPC,NFEQN,1)
      CALL DEFINR('WE  ',MPWE,NFELM,1)
      CALL ZEROR(IA(MPV),NFEQN,1)
      CALL ZEROR(IA(MPC),NFEQN,1)
C
C--4.0 GET NODAL VOLUMETRIC MASS
C
      CALL READV(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--5.0 GET NODAL INITIAL CONCENTRATIONS
C
      CALL READIC(ERR)
      IF(ERR) THEN
        CALL ABORT
        RETURN
      ENDIF
C
C--6.0 OPEN ELEMENT, FLOW AND EXCITATION DATA FILES, & PLOT FILE
C
      OPEN(ND1,FILE=(FNAME(1:LFNAME)//'.FEL'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND ND1

      OPEN(ND2,FILE=(FNAME(1:LFNAME)//'.WDT'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND ND2
      READ(ND2) TWDAT

      OPEN(ND3,FILE=(FNAME(1:LFNAME)//'.EDT'),STATUS='OLD',
     +FORM='UNFORMATTED')
      REWIND ND3
      READ(ND3) TEDAT

      IF(PSCALE.NE.0.0D0) THEN
        CALL NOPEN(ND4,(FNAME(1:LFNAME)//'.PLT'),'FORMATTED')
      ENDIF
C
C--8.0 DEFINE ADDITIONAL SOLUTION ARRAYS
C
      CALL DELETE('FS  ')
      CALL DELETE('CD  ')
      CALL DELETE('CDD ')
      CALL DEFINR('CDD ',MPCDD,NFEQN,1)
      CALL DEFINR('CD  ',MPCD,NFEQN,1)
      CALL DEFINR('FS  ',MPFS,NFEQN,2*NFBAN-1)
      CALL ZEROR(IA(MPCD),NFEQN,1)
      CALL ZEROR(IA(MPCDD),NFEQN,1)
C
C--9.0 CALL PREDIC TO DO THE WORK
C
      CALL PREDIC(IA(MPKEQ),IA(MPF),IA(MPFS),IA(MPV),IA(MPG),IA(MPC),
     +IA(MPCD),IA(MPCDD),TIME,ALPHA,NFNOD,NFEQN,NFBAN,PINT,PSCALE,ERR)

      IF(ERR) CALL ABORT
C
C--10.0 DELETE UNNEEDED ARRAYS & CLOSE FILES
C
      CALL DELETE('FS  ')
      CALL DELETE('CD  ')
      CALL DELETE('CDD ')

      CLOSE(ND1)
      CLOSE(ND2)
      CLOSE(ND3)
      CLOSE(ND4)
C
C--11.0 SKIP TO END-OF-COMMAND DELIMITER 'END'
C
      IF(MODE.EQ.'INTER') RETURN
      IF(MODE.EQ.'BATCH') THEN
 1100   IF(EOC) RETURN
        CALL FREE
        GO TO 1100
      ENDIF
      END

C-----------------------------------------------------------------READIC
      SUBROUTINE READIC(ERR)
C--SUB:READIC - READS & REPORTS INITIAL CONCENTRATION CONDITIONS DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      EXTERNAL ICDAT0

      WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,'   == Initial Conditions: Nodal Concentrations')
      CALL DATGEN(ICDAT0,NFNOD,ERR)
      IF(ERR) RETURN

      CALL REPRTN(IA(MPC),IA(MPKEQ),NFEQN,NFNOD)

      RETURN
      END

C-----------------------------------------------------------------ICDAT0
      SUBROUTINE ICDAT0(N,ERR)
C--SUB:ICDAT0 - CALLS ICDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC

      LOGICAL ERR

      CALL ICDAT1(IA(MPKEQ),IA(MPC),NFNOD,NFEQN,N,ERR)

      RETURN
      END

C-----------------------------------------------------------------ICDAT1
      SUBROUTINE ICDAT1(KEQ,C,NFNOD,NFEQN,N,ERR)
C--SUB:ICDAT1 - READS INITIAL CONCENTRATION CONDITIONS DATA
C
      INCLUDE IOCOM.INC

      INTEGER KEQ(NFNOD)
      REAL*8 C(NFEQN),CDAT
      LOGICAL ERR

      CDAT = 0.0D0
      CALL FREER('C',CDAT,1)
      IF(CDAT.LT.0.0D0) THEN
        WRITE(NTW,2000)
        WRITE(NOT,2000)
        ERR = .TRUE.
        RETURN
      ENDIF
 2000 FORMAT(' **** ERROR: Nodal concentrations may not be negative.')

      NEQ = ABS(KEQ(N))
      C(NEQ) = CDAT

      RETURN
      END

C-----------------------------------------------------------------PREDIC
      SUBROUTINE PREDIC(ID,K,KS,C,E,T,TD,TDD,TIME,ALPHA,NNOD,NEQN,MBAN,
     +PINT,PSCALE,ERR)
C-SUB: PREDIC - PREDICTOR-CORRECTOR 1ST O.D.E. EQUATION SOLVER
C              TIME STEP ESTIMATE BASED ON METHOD IN *HEAT*
C              BY R.L.TAYLOR - U.C. BERKELEY
C     SOLVES EQUATION;
C
C         [K(t)]{T} + [C]{dT/dt} = {E(t)}
C
C      WHERE;   [K(t)]  = STORED IN COMPACT ASYMMETRIC BANDED FORM
C               [C]     = DIAGONAL; STORED AS VECTOR
C               {E(t)}  = EXCITATION; DEFINED PIECE-WISE LINEAR
C
C     BASED ON DIFFERENCE APPROXIMATION;
C
C        {T}n+1 = {T}n + (1-a)DT{dT/dt}n + (a)DT{dT/dt}n+1
C
C     WHERE;   a = "alpha", an integration parameter
C                = 0 corresponds to Forward Difference method
C                = 1 corresponds to Backward Difference method
C                = 1/2 corresponds to Crank-Nicholson method (unstable)
C             DT = time step increment
C
C--- D I C T I O N A R Y   O F   V A R I A B L E S ---------------------
C
C     VARIABLE       DESCRIPTION-----------------------------------------
C-DUMMY
C        ID(NNOD)         : EQUATION NUMBER/CODE (ORDERED BY EQTN #)
C                           EQTN # OF NODE N = ABS(ID(N))
C                           ID(N) = 0    : NODE IS NOT DEFINED DOF
C                           ID(N) = POS  : NODE IS E-PRESCRIBED DOF
C                           ID(N) = NEG  : NODE IS T-PRESCRIBED DOF
C    K(NEQN,2*MBAN-1)    : [K] MATRIX: ASYM-BANDED COMPACT-STORED
C   KS(NEQN,2*MBAN-1)    : [K*] = [C] + aDT[K] MATRIX (SCALED FOR NEG ID)
C        C(NEQN)         : CURRENT [C] (ORDERED BY EQTN #)
C        E(NEQN)         : CURRENT {E} (ORDERED BY EQTN #)
C        T(NEQN)         : CURRENT {T} (ORDERED BY EQTN #)
C        TD(NEQN)        : CURRENT {dT/dt} (ORDERED BY EQTN #)
C        TDD(NEQN)       : INITIAL {d/dt(dT/dt)} TO EST TIME STEP
C        TIME(3)         : START TIME, END TIME, TIME INCREMENT
C        ALPHA           : INTEGRATION PARAMETER
C        NNOD            : NUMBER OF SYSTEM NODES
C        NEQN            : NUMBER OF EQUATIONS
C        MBAN            : HALF BANDWIDTH OF SYSTEM
C        PINT            : OUTPUT RESULTS PRINT INTERVAL
C        PSCALE          : RESULTS PLOT-FILE SCALE FACTOR
C        ERR             : ERROR FLAG
C-----------------------------------------------------------------------

      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE IOCOM.INC
C
C---- PREDIC: DATA & COMMON STORAGE
C
      REAL*8 K(NEQN,2*MBAN-1),KS(NEQN,2*MBAN-1),C(NEQN),E(NEQN),T(NEQN),
     +TD(NEQN),TDD(NEQN),TIME(3),ALPHA,PSCALE
      INTEGER PINT, ID(NNOD)
      LOGICAL ERR, TDOF, KUPDAT, EUPDAT
C
C--1.0 FORM INITIAL [K]
C
      CALL UPDATK(K,TIME(1),KUPDAT,ERR)
      IF(ERR) RETURN
C
C--2.0 COMPUTE INITIAL TEMPERATURE RATES: {dT(0)/dt} FROM
C
C         [C]{dT(0)/dt} = {E(0)} - [K]{T(0)}
C
C---2.1 GET INITIAL EXCITATION
C
      CALL UPDATE(E,TIME(1),EUPDAT,ERR)
      IF(ERR) RETURN
C
C---2.2 FORM RHS: {dT/dt}=0 FOR 'T'-DOF, {E}-[K]{T} FOR 'E'-DOF : SOLVE
C
      DO 22 I=1,NEQN
C-----'T'-DOF: SET {dT/dt}=0
      IF(TDOF(I,ID,NNOD)) THEN
C----- 'T'-DOF: CHECK FOR dT/dt INFINITE
        IF(T(I).NE.E(I)) THEN
          WRITE(NTW,2220) I
          WRITE(NOT,2220) I
 2220     FORMAT(' **** ERROR: Can not compute for step change',
     +    ' in dependent variable number:',I5)
          ERR = .TRUE.
          RETURN
        ELSE
          TD(I) = 0.0D0
        ENDIF
C---- 'E'-DOF: FORM {E}-[K]{T} WHERE [K] IS IN COMPACT STORAGE
      ELSE
        TEMP = E(I)
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 20 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*T(J)
   20   CONTINUE
C----- SOLVE
        TD(I) = TEMP/C(I)
   22 ENDIF
C
C—3.0 COMPUTE TAYLOR'S TIMESTEP CHECK
C
      IF(ECHO) WRITE(NTW,2300)
      WRITE(NOT,2300)
 2300 FORMAT(/,'   — Time Step Estimate for Initial Conditions')
C
C——3.1 COMPUTE INITIAL RATE OF TEMP RATES
C        FORM AND SOLVE: [C]d{dT/dt}/dt = -[K]{dT/dt}
C
      DO 32 I=1,NEQN
      IF(TDOF(I,ID,NNOD)) THEN
        TDD(I) = 0.0D0
      ELSE
        TEMP = 0.0D0
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 30 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*TD(J)
   30   CONTINUE
        TDD(I) = TEMP/C(I)
   32 ENDIF
C
C——3.2 COMPUTE NORMS: ||{T(0)}||, ||{dT(0)/dt}||, ||d/dt{dT(0)/dt}||
C
      TN = 0.0D0
      TDN = 0.0D0
      TDDN = 0.0D0
      DO 34 N=1,NEQN
      TN = TN + T(N)**2
      TDN = TDN + TD(N)**2
   34 TDDN = TDDN + TDD(N)**2
      TN = SQRT(TN)
      TDN = SQRT(TDN)
      TDDN = SQRT(TDDN)
C
C——3.3 EVALUATE TAYLORS EXPRESSION FOR TIME STEP ESTIMATE
C
      B = 0.05D0
      IF(TDDN.NE.0.0D0) THEN
        DTEST = (B*TDN + SQRT(B*B*TDN*TDN + 2.0D0*B*TN*TDDN))/TDDN
        IF(ECHO) WRITE(NTW,2320) B*100.0D0, DTEST,TIME(3)
        WRITE(NOT,2320) B*100.0D0, DTEST,TIME(3)
 2320 FORMAT(/'   — NOTE: Estimated time step to limit error to',
     .' approx.',F5.2,'% is:',G10.3,/
     .'              Specified time step is:',G10.3)
      ELSE
        IF(ECHO) WRITE(NTW,2340)
        WRITE(NOT,2340)
 2340 FORMAT(/'   — NOTE: Unable to estimate time step to limit ',
     .'error for the given system.')
      ENDIF
C
C—4.0 FORM AND FACTOR [K*]
C
      CALL FORMKS(ID,K,KS,C,ALPHA,TIME(3),NNOD,NEQN,MBAN)
      CALL FACTCA(KS,NEQN,MBAN,ERR)
      IF(ERR) RETURN
C
C—5.0 TIME STEP THRU SOLUTION
C
      ADT = ALPHA*TIME(3)
      DTA = (1.0D0 - ALPHA)*TIME(3)
      ISTEP = 0

      DO 500 TM=TIME(1)+TIME(3),TIME(2),TIME(3)
      ISTEP = ISTEP + 1
C
C——5.1 UPDATE [K], FORM AND FACTOR [K*]
C
      CALL UPDATK(K,TM,KUPDAT,ERR)
      IF(ERR) RETURN
      IF(KUPDAT) THEN
        CALL FORMKS(ID,K,KS,C,ALPHA,TIME(3),NNOD,NEQN,MBAN)
        CALL FACTCA(KS,NEQN,MBAN,ERR)
        IF(ERR) RETURN
      ENDIF
C
C——5.2 FORM {E}
C
      CALL UPDATE(E,TM,EUPDAT,ERR)
      IF(ERR) RETURN
C
C——5.3 PREDICT T: {T} = {T} + (1-a)DT{dT/dt}
C
      DO 51 N=1,NEQN
      IF(TDOF(N,ID,NNOD)) THEN
        T(N) = E(N)
      ELSE
        T(N) = T(N) + DTA*TD(N)
      ENDIF
   51 CONTINUE
C
C——5.4 FORM RES:{E}-[K]{T} FOR FLUX-DOF, {dT/dt}*DIAG[K*] FOR TEMP-DOF
C
      CALL RES(ID,T,TD,E,K,KS,NNOD,NEQN,MBAN)
C
C——5.5 SOLVE FOR {dT/dt}
C
      CALL SOLVCA(KS,TD,NEQN,MBAN,ERR)
      IF(ERR) RETURN
C
C——5.6 CORRECT T: {T} = {T} + aDT{dT/dt}
C
      DO 55 N=1,NEQN
      IF(TDOF(N,ID,NNOD)) THEN
        T(N) = E(N)
      ELSE
        T(N) = T(N) + ADT*TD(N)
      ENDIF
   55 CONTINUE
C
C——5.7 REPORT RESULTS
C
      IF(MOD(ISTEP,PINT).EQ.0) THEN
        IF(ECHO) WRITE(NTW,2510) TM
        WRITE(NOT,2510) TM
 2510   FORMAT(/,'   — Response ',46(1H=),' Time: ',G10.3)
        CALL REPRTN(T,ID,NEQN,NNOD)

C————WRITE TO FILE <filename>.PLT for plotting
        IF(PSCALE.NE.0.0D0) THEN
          WRITE(ND4,2530) TM, (CHAR(9),T(I)*PSCALE,I=1,NEQN)
 2530     FORMAT(F10.3,(10(A1,E10.4)))
        ENDIF

      ENDIF

  500 CONTINUE
      RETURN
      END

C—————————————————————————————————————————————————————————————UPDATK
      SUBROUTINE UPDATK(K,TM,KUPDAT,ERR)
C—SUB:UPDATK - UPDATES [K]=[F] IF ELEMENT MASS FLOW RATES CHANGE

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      COMMON /DYNM/ TWDAT,TEDAT
      REAL*8 K(NFEQN,2*NFBAN-1), TM, TWDAT, TEDAT
      LOGICAL ERR, KUPDAT
C
C—1.0 UPDATE ELEMENT FLOW RATES IF(TM.GE.TWDAT)
C
      CALL UPDAT(ND2,TM,TWDAT,IA(MPWE),NFELM,KUPDAT,ERR)
      IF(KUPDAT) THEN
        IF(ECHO) WRITE(NTW,2000) TM
        WRITE(NOT,2000) TM
 2000   FORMAT(/,'   — Element Flow Rate Update ',30(1H-),
     + ' Time: ',G10.3)

        CALL REPRTE(IA(MPWE),NFELM)

        CALL FORMF(IA(MPKEQ),K,IA(MPWE),'BAND',ERR)
      ENDIF
      RETURN
      END

C-----------------------------------------------------------------UPDATE
      SUBROUTINE UPDATE(E,TM,EUPDAT,ERR)
.C--SUB:UPDATE - UPDATES {E}={G} IF EXCITATION CHANGES

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      COMMON /DYNM/ TWDAT,TEDAT
      REAL*8 E(NFEQN), TM, TWDAT, TEDAT
      LOGICAL ERR, EUPDAT

      CALL UPDAT(ND3,TM,TEDAT,E,NFEQN,EUPDAT,ERR)
      IF(EUPDAT) THEN
        IF(ECHO) WRITE(NTW,2000) TM
        WRITE(NOT,2000) TM
 2000   FORMAT(/,'   == Excitation Update ',37(1H-),' Time: ',G10.3)
        CALL REPRTN(E,IA(MPKEQ),NFEQN,NFNOD)
      ENDIF
      RETURN
      END

C------------------------------------------------------------------UPDAT
      SUBROUTINE UPDAT(LUN,T,TD,D,ND,UPDATE,ERR)
C--SUB: UPDAT
C       SEARCHES A SEQUENTIAL DATA RECORD, ON UNIT LUN, OF THE FORM;
C           TD
C           (D(I),I=1,ND)
C           TD
C           (D(I),I=1,ND)
C           ...
C       TO UPDATE DATA VALUES TO CURRENT TIME, "T". IF DATA VALUES ARE
C       UPDATED LOGICAL "UPDATE" IS SET TO TRUE.
C
C         TD         : DISCRETE TIME VALUE
C                    : UPDATED TO NEXT VALUE
C        D(I)        : CORRESPONDING DISCRETE DATA VALUES
C
C       UPDAT MUST BE "PRIMED" BY READING FIRST TD VALUE TO MEMORY
C-----------------------------------------------------------------------

      INCLUDE IOCOM.INC

      REAL*8 D(ND),T,TD
      LOGICAL ERR, UPDATE

      UPDATE = .FALSE.
   10 IF(T.GE.TD) THEN
C----UPDATE DISCRETE DATA VALUES
        READ(LUN, ERR=800, END=900) (D(I),I=1,ND)
        IF(ERR) RETURN
        UPDATE = .TRUE.
C----GET NEXT DISCRETE TIME
        READ(LUN, ERR=800, END=900) TD
        IF(ERR) RETURN
        GO TO 10
      ELSE
        RETURN
      ENDIF

  800 ERR = .TRUE.
      WRITE(NTW,8000)
      WRITE(NOT,8000)
 8000 FORMAT(' **** ERROR: Time history data file read error.')
      RETURN

  900 ERR = .TRUE.
      WRITE(NTW,9000)
      WRITE(NOT,9000)
 9000 FORMAT(
     +' **** ERROR: EOF encountered on time history data file.',/,
     +'             Insufficient time history data.')
      RETURN

      END
```

```
C-----------------------------------------------------------------FORMKS
      SUBROUTINE FORMKS(ID,K,KS,C,ALPHA,DT,NNOD,NEQN,MBAN)
C--SUB:FORMKS - FORMS;
C                       [K*] = [C] + aDT[K]
C
C                SCALES [K*] = [K*]*1.0D15 FOR 'T'-DOF
C-----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)

      REAL*8 K(NEQN,2*MBAN-1),KS(NEQN,2*MBAN-1),C(NEQN)
      INTEGER ID(NNOD)
      LOGICAL TDOF

      ADT = ALPHA*DT
      DO 10 N=1,NEQN
      DO 10 M=1,2*MBAN-1
   10 KS(N,M) =ADT*K(N,M)

      DO 20 N=1,NEQN
   20 KS(N,MBAN) = KS(N,MBAN) + C(N)

      DO 30 N=1,NEQN
   30 IF(TDOF(N,ID,NNOD)) KS(N,MBAN) = KS(N,MBAN)*1.0D15

      RETURN
      END

C--------------------------------------------------------------------RES
      SUBROUTINE RES(ID,T,TD,E,K,KS,NNOD,NEQN,MBAN)
C-SUB:RES - FORMS RHS OF [K*]{dT/dt} = {E*}
C
C            {E*(t)} = {E(t)}-[K]{T(t)}           ; FOR 'E'-DOF
C            {E*(t)} = {dT(t)/dt}*DIAG OF [K*]    ; FOR 'T'-DOF
C
C
C           {E*} IS WRITTEN OVER {TD}
C           [K] & [K*] ARE AYSM-BANDED COMPACT STORED
C-----------------------------------------------------------------------

      IMPLICIT REAL*8 (A-H,O-Z)

      REAL*8 T(NEQN),TD(NEQN),E(NEQN),K(NEQN,2*MBAN-1),
     +KS(NEQN,2*MBAN-1)
      INTEGER ID(NNOD)
      LOGICAL TDOF

      DO 20 I=1,NEQN
C---- SCALE BY DIAGONAL FOR TEMP PRESCRIBED NODES
      IF(TDOF(I,ID,NNOD)) THEN
        TD(I) = TD(I)*KS(I,MBAN)
C---- FORM {E}-[K]{T} WHERE [K] IS IN COMPACT STORAGE
      ELSE
        TEMP = E(I)
        K1 = MAX(1,MBAN-I+1)
        K2 = MIN(2*MBAN-1,MBAN+NEQN-I)
        DO 10 KK=K1,K2
        J = I + KK - MBAN
        TEMP = TEMP - K(I,KK)*T(J)
   10   CONTINUE
        TD(I) = TEMP
      ENDIF
   20 CONTINUE
      RETURN
      END

C-------------------------------------------------------------------TDOF
      FUNCTION TDOF(NEQ,ID,NNOD)
C-FUN:TDOF - DETERMINES IF EQUATION NUMBER NEQ IS A TEMPERATURE DOF
      LOGICAL TDOF
      INTEGER ID(NNOD)
      TDOF = .FALSE.
      DO 10 N=1,NNOD
       IF((ID(N).LT.0).AND.(ABS(ID(N)).EQ.NEQ)) THEN
         TDOF = .TRUE.
         RETURN
       ENDIF
   10 CONTINUE
      RETURN
      END

C----------------------------------------------------------------RESET
      SUBROUTINE RESET
C--SUB:RESET - COMMAND TO RESET CONTAM BY RE-INITIALIZING POINTERS AND
C              COUNTERS AND DELETES ARRAYS LEFT BY CONTAM IN BLANK COMMON
C--HELP LIST-----------------------------------------------------------
C
      CALL INITCN

      RETURN
      END

C--------------------------------------------------------------------C
C                                                                    C
C                   C O N T A M   U T I L I T I E S                  C
C                                                                    C
C--------------------------------------------------------------------C

C----------------------------------------------------------------NDCHK
      SUBROUTINE NDCHK(ND,MAXNUM,NDIM,ERR)
C--SUB:NDCHK - CHECKS FOR OUT-OF-RANGE ELEMENT NODE NUMBERS

      INCLUDE IOCOM.INC

      LOGICAL ERR
      DIMENSION ND(NDIM)
C--- D I C T I O N A R Y   O F   V A R I A B L E S -------------------
C
C    VARIABLE       DESCRIPTION-----------------------------------------
C  INPUT
C    ND             NODE NUMBER ARRAY
C    MAXNUM         LARGEST ALLOWABLE NUMBER
C    NDIM           DIMENSION OF NODE NUMBER ARRAY
C  OUTPUT
C    ERR            ERROR FLAG
C---------------------------------------------------------------------
      DO 10 N=1,NDIM
        NN = ND(N)
        IF(NN.LE.0.OR.NN.GT.MAXNUM) THEN
          WRITE(NTW,2000) NN
          ERR=.TRUE.
        ENDIF
   10 CONTINUE
      RETURN
 2000 FORMAT(' **** ERROR: (Generated) number ',I5,' is out of range.')
      END

C----------------------------------------------------------------READV
      SUBROUTINE READV(ERR)
C--SUB:READV - READS & REPORTS NODE  VOLUME DATA
C
      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      LOGICAL ERR
      EXTERNAL VDAT0

      WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(/,'   -- Nodal Volumetric Mass')
      CALL DATGEN(VDAT0,NFELM,ERR)
      IF(ERR) RETURN

      CALL REPRTN(IA(MPV),IA(MPKEQ),NFEQN,NFNOD)

      RETURN
      END

C----------------------------------------------------------------VDAT0
      SUBROUTINE VDAT0(N,ERR)
C--SUB:VDAT0 - CALLS VDAT1 PASSING ARRAYS
C
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC
```

```
      LOGICAL ERR

      CALL VDAT1(IA(MPKEQ),IA(MPV),NFNOD,NFEQN,N,ERR)

      RETURN
      END

C----------------------------------------------------------------VDAT1
      SUBROUTINE VDAT1(KEQ,V,NFNOD,NFEQN,N,ERR)
C--SUB:VDAT1 - READS NODE VOLUME DATA
C
      INCLUDE IOCOM.INC

      INTEGER KEQ(NFNOD)
      REAL*8 V(NFEQN),VDAT
      LOGICAL ERR

      CALL FREER('V',VDAT,1)
      IF(VDAT.LT.0.0D0) THEN
        WRITE(NTW,2000)
        WRITE(NOT,2000)
        ERR = .TRUE.
        RETURN
      ENDIF
 2000 FORMAT(' **** ERROR: Nodal volumetric mass may not be negative.')
      NEQ = ABS(KEQ(N))
      V(NEQ) = VDAT

      RETURN
      END
C---------------------------------------------------------------DATGEN
      SUBROUTINE DATGEN(DATA0,MAXNO,ERR)
C--SUB:DATGEN - READS AND GENERATES DATA BY INCREMENTING RULE;
C                 n1,n2,n3 = FIRST #, LAST #, INCREMENT
C              GIVEN DATA LINE OF FORM
C                 n1,n2,n3  D1=n4,n5,... D2=n6,n7,... etc.
C              CALLS SUBROUTINE "DATA0" TO READ DATA (D1, D2, etc.)
C              RETURNS WHEN DATA LINE IS BLANK, IS ";", OR IS "END"
C              CHECKS ALL GENERATED NUMBERS .LE. MAXNO FOR MAXNO.GT.0
C
      INCLUDE IOCOM.INC

      LOGICAL ERR, FIRSTL
      INTEGER IJK(3)
      EXTERNAL DATA0

      ERR = .FALSE.
      FIRSTL = .TRUE.
C
C--1.0 GET LINE OF DATA
C
  100 IF(MODE.EQ.'INTER') CALL PROMPT(' DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREENR(NTW)
C
C--2.0 CHECK FOR "END"
C
      IF(EOC) THEN
        IF(FIRSTL) THEN
          WRITE(NTW,2200)
          WRITE(NOT,2200)
 2200     FORMAT(' **** ERROR: Data expected; "END" found.')
          ERR = .TRUE.
          RETURN
        ELSE
          RETURN
        ENDIF
      ENDIF
C
C--3.0 GET INCREMENTING RULE; RETURN IF IJK(1).EQ.0
C
      IJK(1) = 0
      CALL FREEI(' ',IJK(1),3)
      IF(IJK(1).EQ.0) RETURN
      IF(IJK(2).EQ.0) IJK(2)=IJK(1)
      IF(IJK(3).EQ.0) IJK(3)=1
      DO 300 N=IJK(1), IJK(2), IJK(3)
        IF(MAXNO.GT.0) CALL NDCHK(N,MAXNO,1,ERR)
        IF(ERR) RETURN
        CALL DATA0(N,ERR)
      IF(ERR) RETURN
  300 CONTINUE

      FIRSTL = .FALSE.
      GO TO 100

      END

C-------------------------------------------------------------------ELGEN
      SUBROUTINE ELGEN(ELEMO,KEQ,NELDOF,NSYNOD,MSYBAN,ERR)
C--SUB:ELEMIN - READS ELEMENT NUMBER, CONNECTIVITY, & GENERATION DATA
C               GENERATES MISSING ELEMENTS, UPDATES SYSTEM BANDWIDTH
C               CALLS "ELEMO" TO READ ELEMENT PROPERTY DATA
C               RETURNS WHEN DATA LINE IS BLANK, IS ":", OR IS "END"
C               CHECKS ALL GENERATED NODE NUMBERS .LE. NSYNOD
C
C               ** CURRENTLY LIMITED TO FOUR-NODE ELEMENTS OR LESS **
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S ----------------
C
C     VARIABLE      DESCRIPTION-----------------------------------------
C  INPUT
C     ELEMO         PROCEDURE NAME TO READ ELEMENT PROPERTY DATA
C     NELDOF        NUMBER OF ELEMENT DEGREES OF FREEDOM
C     KEQ           SYSTEM EQUATION NUMBERS (BY NODE NUMBER)
C     NSYNOD        NUMBER OF SYSTEM NODES
C  OUTPUT
C     MSYBAN        SYSTEM BAND WIDTH
C     ERR           ERROR FLAG
C  LOCAL
C     LMNEW,LMOLD   ELEMENT LOCATION/CONNECTIVITY DATA
C     NOLD,NNEW     ELEMENT NUMBERS
C     INCR          GENERATION INCREMENT
C-----------------------------------------------------------------------

      INCLUDE IOCOM.INC

      LOGICAL ERR
      INTEGER NELDOF,LMNEW(4),LMOLD(4),NOLD,NNEW,KEQ(NSYNOD)
      EXTERNAL ELEMO
C
C--1.0 GET FIRST LINE OF ELEMENT DATA
C
      INCR = 0
      IF(MODE.EQ.'INTER') CALL PROMPT(' DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
C---- CHECK FOR "END"
      IF(EOC) RETURN
      NOLD = 0
      CALL FREEI(' ',NOLD,1)
      IF(NOLD.EQ.0) RETURN
      CALL FREEI('I',LMOLD(1),NELDOF)

      CALL NDCBK(LMOLD,NSYNOD,NELDOF,ERR)
      IF(ERR) RETURN
      CALL ELBAN(KEQ,LMOLD,MSYBAN,NELDOF,NSYNOD)
      CALL ELEMO(NOLD,LMOLD,ERR)
      IF(ERR) RETURN
C
C--2.0 GET NEXT LINE OF ELEMENT DATA
C
   20 IF(MODE.EQ.'INTER') CALL PROMPT(' DATA>')
      CALL FREE
      IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
C---- CHECK FOR "END"
      IF(EOC) RETURN
C---- GET NEW ELEMENT INFORMATION
      NNEW = 0
      CALL FREEI(' ',NNEW,1)
      IF(NNEW.EQ.0) RETURN
      CALL FREEI('I',LMNEW(1),NELDOF)
      CALL FREEI('N',INCR,1)
      IF(INCR.EQ.0) INCR=1
C---- CHECK NUMERICAL ORDER
      IF(NNEW.LE.NOLD) THEN
        WRITE(NTW,2200) NNEW
        WRITE(NOT,2200) NNEW
        ERR=.TRUE.
        RETURN
      ENDIF
C---- GENERATE MISSING ELEMENTS
      IF(NNEW.GT.NOLD+1) THEN
        DO 24 N=NOLD+1,NNEW-1,1
         DO 22 I=1,NELDOF
   22    LMOLD(I) = LMOLD(I) + INCR
        CALL NDCBK(LMOLD,NSYNOD,NELDOF,ERR)
        IF(ERR) RETURN
        CALL ELEMO(N,LMOLD,ERR)
        IF(ERR) RETURN
   24  CONTINUE
      ENDIF
C---- DO NEW ELEMENT
      NOLD = NNEW
      DO 26 I=1,NELDOF
   26 LMOLD(I) = LMNEW(I)

      CALL NDCBK(LMOLD,NSYNOD,NELDOF,ERR)
      IF(ERR) RETURN
      CALL ELBAN(KEQ,LMOLD,MSYBAN,NELDOF,NSYNOD)
      CALL ELEMO(NOLD,LMOLD,ERR)
      IF(ERR) RETURN

      GO TO 20

 2200 FORMAT(' **** ERROR: Element number ',I5,' is out of order.')
      END

C-------------------------------------------------------------------ELBAN
      SUBROUTINE ELBAN( KEQ,LM,MSYBAN,NELDOF,NSYNOD)
C--SUB:ELBAN - COMPUTES ELEMENT BANWIDTH & UPDATES SYSTEM BANDWIDTH

      DIMENSION LM(NELDOF),KEQ(NSYNOD)

C---- D I C T I O N A R Y   O F   V A R I A B L E S ----------------
C
C     VARIABLE      DESCRIPTION-----------------------------------------
C  INPUT
C     LM            ELEMENT LOCATION/CONNECTIVITY ARRAY
C     NELDOF        NUMBER OF ELEMENT DEGREES OF FREEDOM
C     MSYBAN        CURRENT SYSTEM BANDWIDTH
C     KEQ           SYSTEM EQUATION NUMBERS (BY NODE NUMBER)
C     NSYNOD        NUMBER OF SYSTEM NODES
C  OUTPUT
C     MSYBAN        UPDATED SYSTEM BAND WIDTH
C-----------------------------------------------------------------------
      MAX = ABS(KEQ(LM(1)))
      MIN = ABS(KEQ(LM(2)))
      DO 10 I=1,NELDOF
        NN = ABS(KEQ(LM(I)))
        IF(NN.GT.MAX) MAX=NN
        IF(NN.LT.MIN) MIN=NN
   10 CONTINUE
      MELBAN = MAX-MIN+1
      IF(MELBAN.GT.MSYBAN) MSYBAN=MELBAN
      RETURN
      END

C------------------------------------------------------------------REPRTN
      SUBROUTINE REPRTN(X,KEQ,NX,NKEQ)
C--SUB:REPRTN - REPORTS VECTOR (X)IN NODE ORDER SEQUENCE
C               X(NX) = VECTOR OF VALUES ORDERED BY EQUATION NUMBER
C               KEQ(NKEQ) = EQUATION NUMBERS ORDERED BY NODE NUMBER
C                  NEG = INDEPENDENT DOF
C                   0  = UNDEFINED DOF
C                  POS = DEPENDENT DOF
C               INDEPENDENT DOFS ARE FLAGGED WITH A '*'
C               UNDEFINED DOF ARE FLAGED WITH A "U"

      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC

      REAL*8 X(NX),XX(5)
      INTEGER KEQ(NKEQ)
      CHARACTER*1 FLG(5)

      WRITE(NOT,2000)
      IF(ECHO) WRITE(NTW,2000)
 2000 FORMAT(/,
     .13X,'"*" = independent DOFs        "U" = undefined DOFs.',//,
     .6X,4(2X,'Node    Value',3X))

      DO 100 N=1,NKEQ,4
        MM = MIN(N+3,NKEQ)
        DO 10 I=N,MM,1
          NEQ = KEQ(I)
          MNEQ = ABS(NEQ)
          IF(NEQ.LT.0) THEN
            XX(I-N+1) = X(MNEQ)
            FLG(I-N+1) = '*'
          ELSEIF(NEQ.EQ.0) THEN
            XX(I-N+1) = 0.0D0
            FLG(I-N+1) = 'U'
          ELSE
            XX(I-N+1) = X(MNEQ)
            FLG(I-N+1) = ' '
          ENDIF
   10   CONTINUE
        IF(ECHO) WRITE(NTW,2010) (I,FLG(I-N+1),XX(I-N+1),I=N,MM)
  100   WRITE(NOT,2010) (I,FLG(I-N+1),XX(I-N+1),I=N,MM)

 2010 FORMAT((6X,4(I6,1A1,G11.3)))

      RETURN
      END

C-----------------------------------------------------------------REPRTE
      SUBROUTINE REPRTE(X,NX)
C--SUB:REPRTE - REPORTS VECTOR(X)IN ELEMENT ORDER SEQUENCE
C               X(NX) = VECTOR OF VALUES ORDERED BY ELEMENT NUMBER
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC

      DIMENSION X(NX)

        WRITE(NOT,2000)
        IF(ECHO) WRITE(NTW,2000)
        WRITE(NOT,2010) (N, X(N), N=1,NX)
        IF(ECHO) WRITE(NTW,2010) (N, X(N), N=1,NX)

 2000 FORMAT(/,6X,4(2X,'Elem    Value',3X))
 2010 FORMAT((6X,4(I6,1X,G11.3)))

      RETURN
      END

C------------------------------------------------------------------FORMF
      SUBROUTINE FORMF(KEQ,F,WE,FORM,ERR)
C--SUB:FORMF - CALLS FORMFO TO FORM SYSTEM FLOW MATRIX
C              ARRAY CONT USED TO CHECK NODAL MASS FLOW CONTINUITY
      COMMON MTOT,NP,IA(1)

      INCLUDE CNTCOM86.INC

      REAL*8 F(NFEQN,1), WE(NFELM)
      INTEGER KEQ(NFNOD), MPCONT
      LOGICAL ERR
      CHARACTER FORM*4

      CALL DELETE('CONT')
      CALL DEFINR('CONT',MPCONT,NFEQN,1)
      CALL ZEROR(IA(MPCONT),NFEQN,1)

      IF(FORM.EQ.'BAND') CALL ZEROR(F,NFEQN,2*NFBAN-1)
      IF(FORM.EQ.'FULL') CALL ZEROR(F,NFEQN,NFEQN)
      CALL FORMFO(KEQ,F,WE,IA(MPCONT),FORM,ERR)

      CALL DELETE('CONT')

      RETURN
      END

C-----------------------------------------------------------------FORMFO
      SUBROUTINE FORMFO(KEQ,F,WE,CONT,FORM,ERR)
C--SUB:FORMFO - FORMS SYSTEM FLOW MATRIX
C               ARRAY CONT USED TO CHECK NODAL MASS FLOW CONTINUITY
```

```
      IMPLICIT REAL*8(A-H,O-Z)

      INCLUDE IOCOM.INC
      INCLUDE CNTCOM86.INC

      REAL*8 F(NFEQN,1), WE(NFELM), ELF(2,2), CONT(NFEQN),EFF
      INTEGER KEQ(NFNOD), LM(2)
      LOGICAL ERR
      CHARACTER FORM*4
C
C--1.0 FOR EACH ELEMENT FORM ELEMENT FLOW MATRIX AND ADD TO [F]
C      ACCUMULATE TOTAL MASS FLOW (CONTINUITY) AT EACH NODE

      REWIND ND1
      DO 10 N=1,NFELM
      READ(ND1, ERR=900, END=900) LM(1),LM(2),EFF
      W = WE(N)
      N1 = ABS(KEQ(LM(1)))
      N2 = ABS(KEQ(LM(2)))
      IF(W.GT.0.0D0) THEN
        ELF(1,1) = W
        ELF(1,2) = 0.0D0
        ELF(2,1) = -W*(1.0D0-EFF)
        ELF(2,2) = 0.0D0
        CONT(N1) = CONT(N1) + W
        CONT(N2) = CONT(N2) - W
      ELSEIF(W.LT.0.0D0) THEN
        ELF(1,1) = 0.0D0
        ELF(1,2) = W*(1.0D0-EFF)
        ELF(2,1) = 0.0D0
        ELF(2,2) = -W
        CONT(N1) = CONT(N1) + W
        CONT(N2) = CONT(N2) - W
      ELSE
        GO TO 10
      ENDIF
      IF (FORM.EQ.'BAND') CALL ADDCA(KEQ,NFNOD,ELF,F,2,NFEQN,NFBAN,LM)
      IF (FORM.EQ.'FULL') CALL ADDA(KEQ,NFNOD,ELF,F,2,NFEQN,LM)
   10 CONTINUE
C
C--2.0 REPORT NET TOTAL MASS FLOW
C
      WRITE(NOT,2200)
      IF(ECHO) WRITE(NTW,2200)
 2200 FORMAT(/,'   -- Net Total Mass Flow')
      CALL REPRTN(CONT,KEQ,NFEQN,NFNOD)

      RETURN

  900 WRITE(NTW,2900)
      WRITE(NOT,2900)
 2900 FORMAT(
     +' **** ERROR: Read or EOF error on flow element data file')
      ERR = .TRUE.
      RETURN

      END

C------------------------------------------------------------------ADDCA
      SUBROUTINE ADDCA(KEQ,NSYNOD,ELA,SYA,NELDOF,NSYDOF,MSYBAN,LM)
C--SUB:ADDCA - ADDS ELEMENT ARRAY TO COMPACT ASYMMETRIC SYSTEM ARRAY
C
      REAL*8 ELA(NELDOF,NELDOF), SYA(NSYDOF,1)
      INTEGER KEQ(NSYNOD),LM(NELDOF)
C
C-- D I C T I O N A R Y   O F   V A R I A B L E S ---------------------
C
C     VARIABLE                 DESCRIPTION-----------------------------
C     KEQ(NSYNOD)            : SYSTEM NODAL EQUATION NUMBERS
C     NSYNOD                 : NUMBER OF SYSTEM NODES
C     ELA(NELDOF,NELDOF)     : ELEMENT ARRAY
C     SYA(NSYDOF,2*MSYBAN-1) : COMPACTED ASYM. SYSTEM ARRAY
C     NELDOF                 : NUMBER OF ELEMENT DEGREES OF FREEDOM
C     NSYDOF                 : NUMBER OF SYSTEM DEGREES OF FREEDOM
C     MSYBAN                 : HALF BANDWIDTH OF SYSTEM ARRAY
C     LM(NELDOF)             : ELEMENT LOCATION/CONNECTIVITY
C
C----------------------------------------------------------------------
      DO 20 I=1,NELDOF
      II = ABS(KEQ(LM(I)))
      DO 10 J=1,NELDOF
        JJ = MSYBAN - II + ABS(KEQ(LM(J)))
        SYA(II,JJ) = SYA(II,JJ) + ELA(I,J)
   10   CONTINUE
   20 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------------ADDA
      SUBROUTINE ADDA(KEQ,NSYNOD,ELA,SYA,NELDOF,NSYDOF,LM)
C--SUB:ADOCA - ADDS ELEMENT ARRAY TO FULL ASYMMETRIC SYSTEM ARRAY
C
      REAL*8 ELA(NELDOF,NELDOF), SYA(NSYDOF,1)
      INTEGER KEQ(NSYNOD),LM(NELDOF)
C
C---- D I C T I O N A R Y   O F   V A R I A B L E S -----------------
C
C     VARIABLE                DESCRIPTION----------------------------
C     KEQ(NSYNOD)           : SYSTEM NODAL EQUATION NUMBERS
C     NSYNOD                : NUMBER OF SYSTEM NODES
C     ELA(NELDOF,NELDOF)    : ELEMENT ARRAY
C     SYA(NSYDOF,2*MSYBAN-1) : COMPACTED ASYM. SYSTEM ARRAY
C     NELDOF                : NUMBER OF ELEMENT DEGREES OF FREEDOM
C     NSYDOF                : NUMBER OF SYSTEM DEGREES OF FREEDOM
C     MSYBAN                : HALF BANDWIDTH OF SYSTEM ARRAY
C     LM(NELDOF)            : ELEMENT LOCATION/CONNECTIVITY
C
C--------------------------------------------------------------------
      DO 20 I=1,NELDOF
        II = ABS(KEQ(LM(I)))
        DO 10 J=1,NELDOF
          JJ = ABS(KEQ(LM(J)))
          SYA(II,JJ) = SYA(II,JJ) + ELA(I,J)
   10   CONTINUE
   20 CONTINUE
      RETURN
      END

C--------------------------------------------------------------------C
C                                                                    C
C      C O M M A N D   P R O C E S S O R   U T I L I T I E S         C
C                                                                    C
C--------------------------------------------------------------------C

C-----------------------------------------------------------------NOPEN
      SUBROUTINE NOPEN(LUN,FNAME,FRM)
C--SUB: NOPEN - OPENS A FILE AS A NEW FILE WHETHER IT EXISTS OR NOT
C                  LUN - LOGICAL UNIT NUMBER
C                  FNAME - FILENAME
C                  FRM = FORM; 'UNFORMATTED' OR 'FORMATTED'

      INTEGER LUN
      CHARACTER FNAME*(*), FRM*(*)
      LOGICAL FOUND

      INQUIRE(FILE=FNAME,EXIST=FOUND)
      IF(FOUND) THEN
           OPEN(LUN,FILE=FNAME,STATUS='OLD',FORM=FRM)
           IF(FRM.EQ.'FORMATTED') THEN
         WRITE(LUN,2000) LUN
 2000    FORMAT(I6)
        ELSEIF(FRM.EQ.'UNFORMATTED') THEN
             WRITE(LUN) LUN
           ENDIF
           CLOSE(LUN,STATUS='DELETE')
           OPEN(LUN,FILE=FNAME,STATUS='NEW',FORM=FRM)
      ELSE
        OPEN(LUN,FILE=FNAME,STATUS='NEW',FORM=FRM)
      ENDIF

      RETURN
      END

C----------------------------------------------------------------PROMPT
      SUBROUTINE PROMPT(STRING)
C--SUB:PROMPT - INLINE PROMPT
      INCLUDE IOCOM.INC
      CHARACTER STRING*(*)
      WRITE(NTW,'(A,\)') STRING
      RETURN
      END

C------------------------------------------------------------------PROMS
      SUBROUTINE PROMS(N)
C--SUB:PROMS - "HOLLERITH PROMPT"

      COMMON MTOT,NP,IA(1)

      INCLUDE IOCOM.INC
      CHARACTER*1 NCMND,M
      COMMON /CMND/ NCMND(8),M(4,7)

C-----PROMPT FOR ARRAY NAMES
      IF(MODE.EQ.'BATCH') GO TO 900
      DO 200 I=1,N
  100 IF(M(1,N).NE.' ') GO TO 200
      WRITE(NTW,2000) N
      CALL FREE
      CALL FREEC(' ',M(1,N),8,1)
      GO TO 100
  200 CONTINUE
C
  900 RETURN

 2000 FORMAT('   ** Enter array name "',1I1,'": ')
      END

C------------------------------------------------------------------PROMI
      SUBROUTINE PROMI(NR,NC)
C--SUB: PROMI - "INTEGER PROMPT"

      INCLUDE IOCOM.INC

C-----ASK FOR NUMBER OF ROWS AND COLUMNS
      IF(MODE.EQ.'BATCH') GO TO 900
  100 IF(NR.GT.0) GO TO 200
      CALL PROMPT('   ** Enter number of rows: ')
      CALL FREE
      CALL FREEI(' ',NR,1)
      GO TO 100
C
  200 IF(NC.GT.0) GO TO 900
      CALL PROMPT('   ** Enter number of columns: ')
      CALL FREE
      CALL FREEI(' ',NC,1)
      GO TO 200
C
  900 RETURN
      END

C------------------------------------------------------------------ABORT
      SUBROUTINE ABORT
C--SUB:ABORT - ABORTS COMMAND AND RETURNS TO INTERACTIVE MODE
C-----------------------------------------------------------------------
      INCLUDE IOCOM.INC

      WRITE(NTW,2000)
      WRITE(NOT,2000)
 2000 FORMAT(' **** COMMAND ABORTED')
      IF(MODE.EQ.'BATCH') CALL RETRN

      RETURN
      END

C-----------------------------------------------------------------------
C                                                                      C
C                  C A L S A P X   L I B R A R Y                       C
C                                                                      C
C         AN EXTENSION OF "CAL-SAP" LIBRARY OF SUBROUTINES             C
C                DEVELOPED BY ED WILSON, U.C. BERKELEY                 C
C-----------------------------------------------------------------------
C
C 1.0 FREE-FIELD INPUT SUBROUTINES
C
C-------------------------------------------------------------------FREE
```

```
      SUBROUTINE FREE
C--SUB:FREE - READ LINE OF FREE FIELD DATA
C             COMMENTS LINES ECHOED TO SCREEN

      INCLUDE IOCOM.INC
      INCLUDE FRECOM.INC
C
C-0.0-INITIALIZE VARIABLES
C
      EOD = .FALSE.
      EOC = .FALSE.
      DO 5 I=1,160
    5 LINE(I)=' '
C
C-1.0 GET LINE OF DATA
C
   10 I = 1
      II= 80
      READ(NCMD,1000,ERR=100) (LINE(K),K=I,II)

C-----CHECK FOR ADDITIONAL LINE

      JJ = LENTRM(LLINE)
      DO 12 K=I,JJ
        IF(LINE(K).EQ.'\') THEN
          I = K
          II= K+79
          READ(NCMD,1000,ERR=100) (LINE(KK),KK=I,II)
 1000     FORMAT(80A1)
          GO TO 14
        ENDIF
   12 CONTINUE

C-----CHECK FOR COMMENT

   14 IF(LINE(1).EQ.'*') THEN
        IF(MODE.EQ.'BATCH') CALL FREEWR(NTW)
        CALL FREEWR(NOT)
        GO TO 10
      ENDIF
C
C-2.0 DETERMINE LENGTH-OF-INFORMATION
C
      JJ = LENTRM(LLINE)
C
C-3.0 DETERMINE LENGTH-OF-DATA AND CONVERT DATA TO UPPER CASE
C
      ISP = ICHAR(' ')
      IA  = ICHAR('a')
      DO 30 I=1,JJ
        IF(LINE(I).EQ.'<') GO TO 32
        NN = ICHAR(LINE(I))
        IF(NN.GE.IA) LINE(I) = CHAR(NN-ISP)
   30 CONTINUE
   32 II = I - 1
C
C-4.0 CHECK FOR END-OF-DATAGROUP & END-OF-COMMAND
C
      IF(LINE(1).EQ.'<') EOD = .TRUE.

      IF(LINE(1)//LINE(2)//LINE(3).EQ.'END') EOC = .TRUE.

      RETURN

C-----ERROR IN READ -----

  100 WRITE(NOT,2000)
      WRITE(NTW,2000)
 2000 FORMAT(' **** ERROR: Error in reading input line.')
      CALL ABORT

      END

C------------------------------------------------------------------FREEWR
      SUBROUTINE FREEWR(LUN)
C--SUB:FREEWR - WRITE COMMAND/DATA LINE TO FILE LUN
C               LUN = LOGICAL UNIT NUMBER TO WRITE TO

      INCLUDE IOCOM.INC
      INCLUDE FRECOM.INC

      WRITE(LUN,2000) (LINE(I),I=1,JJ)
 2000 FORMAT (1X,80A1)

      RETURN
      END

C------------------------------------------------------------------FREEFN
      SUBROUTINE FREEFN(SEP,NC,FOUND)
C--SUB:FREEFN - FINDS NEXT NC-CHARACTER SEPARATOR IN INPUT FILE
C               SEP(NC)*1 = CHARACTER STRING

      INCLUDE IOCOM.INC
      INCLUDE FRECOM.INC

      CHARACTER*1 SEP(NC)
      LOGICAL FOUND

      FOUND = .FALSE.

   50 CALL FREE
      IF(NC.LE.II) THEN
        DO 60 N=1,NC
   60   IF(SEP(N).NE.LINE(N)) GO TO 50
        FOUND = .TRUE.
        RETURN
      ELSE
        GO TO 50
      ENDIF

      RETURN
      END

C------------------------------------------------------------------ FREER
      SUBROUTINE FREER(IC,DATA,NUM)
C--SUB:FREER - FIND AND INTERPRET REAL DATA
C              IC*1 = DATA IDENTIFIER CHARACTER
C              DATA = REAL DATA RETURNED
C              NUM  = NUMBER OF DATA VALUES TO EXTRACT

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION DATA(10)
      CHARACTER IC*1

      INCLUDE FRECOM.INC

C-----FIND REAL STRING ----------------------------------
   90 I=0
      IF(IC.EQ.' ') GO TO 250
      DO 100 I=1,II
      IF((LINE(I).EQ.IC).AND.(LINE(I+1).EQ.'=')) GO TO 250
  100 CONTINUE
      RETURN
C---- EXTRACT REAL DATA -----
  250 DO 260 J=1,NUM
  260 DATA(J)=0.0
      DO 300 J=1,NUM
      JJ=0
  270 IF(I.GT.II) GO TO 300
      CALL FREER1(I,XX,NN)
      IF(JJ.NE.0) GO TO 275
      DATA(J) = XX
      GO TO 290
C-----ARITHMETRIC STATEMENT -----
  275 IF(JJ.EQ.1) DATA(J)=DATA(J)*XX
      IF(JJ.EQ.2) DATA(J)=DATA(J)/XX
      IF(JJ.EQ.3) DATA(J)=DATA(J)+XX
      IF(JJ.EQ.4) DATA(J)=DATA(J)-XX
      IF(JJ.NE.5) GO TO 290
C-----EXPONENTIAL DATA -----
      JJ = DABS(XX)
      IF(JJ.EQ.0) GO TO 290
      DO 280 K=1,JJ
      IF(XX.LT.0.0) DATA(J) = DATA(J)/10.
      IF(XX.GT.0.0) DATA(J) = DATA(J)*10.
  280 CONTINUE
C-----SET TYPE OF STATEMENT -----
  290 JJ=0
      IF(LINE(I).EQ.'*') JJ=1
      IF(LINE(I).EQ.'/') JJ=2
      IF(LINE(I).EQ.'+') JJ=3
      IF(LINE(I).EQ.'-') JJ=4
      IF(LINE(I).EQ.'E') JJ=5
      IF(LINE(I+1).EQ.'=') JJ=0
      IF(JJ.NE.0) GO TO 270
      IF(MN.GT.9) RETURN
  300 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------------FREER1
      SUBROUTINE  FREER1(I,XX,MN)
C-SUB:FREER1 - INTERPRETS A SINGLE REAL VALUE

      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE FRECOM.INC

C------CONVERT STRING TO REAL FLOATING POINT NUMBER --
      IF(LINE(I+1).EQ.'=') I=I+1
      Y=0
      IS=1
      XX=0.0
      IF(LINE(I+1).EQ.'-') THEN
        IS=-1
        I=I+1
      ELSEIF(LINE(I+1).EQ.'+') THEN
        IS=1
        I=I+1
      ELSE
        CONTINUE
      ENDIF
  267 IF(LINE(I+1).NE.' ') GO TO 270
      I=I+1
      IF(I.GT.II) GO TO 300
      GO TO 267
  270 I=I+1
      IF(I.GT.II) GO TO 300
      IF((LINE(I).EQ.' ').AND.(LINE(I+1).EQ.' ')) GO TO 270
      MN = ICHAR( LINE(I) ) - ICHAR('0')
      XN=ISIGN(MN,IS)
      IF(LINE(I).NE.'.') GO TO 275
      Y=1.0
      GO TO 270
  275 IF(LINE(I).EQ.' ') GO TO 300
      IF(LINE(I).EQ.',') GO TO 300
      IF((MN.LT.0).OR.(MN.GT.9)) GO TO 300
      IF(Y.EQ.0) GO TO 280
      Y=Y/10.
      XN=XN*Y
      XX=XX+XN
      GO TO 270
  280 XX=10.*XX+XN
      GO TO 270
  300 RETURN
      END

C---------------------------------------------------------------------- FREEI
      SUBROUTINE FREEI(IC,IDATA,NUM)
C--SUB:FREEI - FIND AND INTERPRET INTEGER DATA
C             IC*1  = DATA IDENTIFIER CHARACTER
C             IDATA = INATEGER DATA RETURNED
C             NUM   = NUMBER OF DATA VALUES TO EXTRACT
      CHARACTER*1 IC,LNE
      DIMENSION IDATA(72)

      INCLUDE FRECOM.INC

C------FIND INTEGER STRING ----------------------
   90 I=0
      IF(IC.EQ.' ') GO TO 200
      DO 100 I=1,II
      IF((LINE(I).EQ.IC).AND.(LINE(I+1).EQ.'=')) GO TO 200
  100 CONTINUE
      RETURN
C------ZERO INTEGER STRING -----
  200 DO 210 J=1,NUM
  210 IDATA(J)=0
      IF(LINE(I+1).EQ.'=') I=I+1
      DO 250 J=1,NUM
      ISIGN = 1
C------SKIP BLANKS BETWEEN INTEGERS -----
  215 IF(LINE(I+1).NE.' ') GO TO 220
      I=I+1
      IF(I.GT.II) GO TO 900
      GO TO 215
  220 I=I+1
      IF(I.GT.II) GO TO 230
C------CHECK FOR SIGN -----
      LNE = LINE(I)
      IF(LNE.NE.'-') GO TO 225
      ISIGN = -1
      GO TO 220
C------EXTRACT INTEGER -----
  225 IF(LNE.EQ.' ') GO TO 230
      IF(LNE.EQ.',') GO TO 230
      IF(LNE.EQ.':') GO TO 230
      MN = ICHAR(LNE) - ICHAR('0')
      IF((MN.LT.0).OR.(MN.GT.9)) GO TO 900
      IDATA(J)=10*IDATA(J)+MN
      GO TO 220
C------SET SIGN -----
  230 IDATA(J) = IDATA(J)*ISIGN
  250 CONTINUE
  900 RETURN
      END

C---------------------------------------------------------------------- FREEC
      SUBROUTINE FREEC(IC,IDATA,NC,NUM)
C--SUB:FREEC - FIND AND INTERPRET CHARACTER DATA
C             IC*1  = DATA IDENTIFIER CHARACTER
C             IDATA = CHARACTER DATA RETURNED
C             NC    = NUMBER OF CHARACTERS PER DATA VALUE
C             NUM   = NUMBER OF DATA VALUES TO EXTRACT
      CHARACTER*1 IC,IDATA
      DIMENSION IDATA(NC,NUM)

      INCLUDE FRECOM.INC

C------FIND DATA IDENTIFIER
   90 I=0
      IF(IC.EQ.' ') GO TO 200
      DO 100 I=2,II
      IF((LINE(I-1).EQ.IC).AND.(LINE(I).EQ.'=')) GO TO 200
  100 CONTINUE
      RETURN
C------EXTRACT CHARACTER DATA -----
  200 DO 210 J=1,NUM
      DO 210 N=1,NC
  210 IDATA(N,J)=' '
C
      DO 300 J=1,NUM
  260 I = I + 1
      IF(I.GT.II) GO TO 400
      IF(LINE(I).EQ.',') GO TO 260
      IF(LINE(I).EQ.' ') GO TO 260
      DO 290 N=1,NC
      IF(LINE(I).EQ.':') GO TO 300
      IF(LINE(I).EQ.' ') GO TO 300
      IF(LINE(I).EQ.',') GO TO 300
      IDATA(N,J) = LINE(I)
      IF(N.EQ.NC) GO TO 290
      I = I + 1
  290 CONTINUE
  300 CONTINUE
  400 RETURN
      END

C----------------------------------------------------------------------LENTRM
      FUNCTION LENTRM(STRING)
C--FUN:LENTRM - DETERMINES LENGTH OF TRIMMED STRING - A STRING WITH
C               TRAILING BLANKS REMOVED
C
C       LENTOT  : THE TOTAL LENGTH OF THE STRING
C       LENTRM  : THE LENGTH OF THE TRIMMED STRING
C-----------------------------------------------------------------------------
      CHARACTER STRING*(*)
      INTEGER LENTOT, LENTRM
      LENTOT = LEN(STRING)

      DO 10 I=LENTOT,1,-1
        IF(STRING(I:I).NE.' ') GO TO 20
   10 CONTINUE

   20 LENTRM = I

      RETURN
      END

C
C 2.0 DYNAMIC ARRAY MANAGEMENT
C

C------------------------------------------------------------------ DEFINR
      SUBROUTINE DEFINR(NAME,NA,NR,NC)
C--SUB:DEFINR - DEFINE DIRECTORY AND RESERVE STORAGE
C               FOR REAL ARRAY IN DATABASE
C               NAME - NAME OF ARRAY
C               NA   - BLANK COMMON POINTER TO ARRAY (RETURNED)
C               NR   - NUMBER OF ROWS
C               NC   - NUMBER OF COLUMNS
C-------------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      CHARACTER*1 NAME(4)
      NP = 2
      CALL DEFIN(NAME,NA,NR,NC)
      RETURN
      END

C------------------------------------------------------------------ DEFINI
      SUBROUTINE DEFINI(NAME,NA,NR,NC)
C--SUB:DEFINI - DEFINE DIRECTORY AND RESERVE STORAGE
C               FOR INTEGER ARRAY IN DATABASE
C               NAME - NAME OF ARRAY
C               NA   - BLANK COMMON POINTER TO ARRAY (RETURNED)
C               NR   - NUMBER OF ROWS
C               NC   - NUMBER OF COLUMNS
C-------------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      CHARACTER*1 NAME(4)
      NP = 1
      CALL DEFIN(NAME,NA,NR,NC)
      RETURN
      END

C------------------------------------------------------------------ DEFINC
      SUBROUTINE DEFINC(NAME,NA,NR,NC)
C--SUB:DEFINC - DEFINE DIRECTORY AND RESERVE STORAGE
C               FOR CHARACTER*1 (HOLLERITH)  ARRAY IN DATABASE
C               NAME - NAME OF ARRAY
C               NA   - BLANK COMMON POINTER TO ARRAY (RETURNED)
C               NR   - NUMBER OF ROWS
C               NC   - NUMBER OF COLUMNS
C-------------------------------------------------------------------------
      CHARACTER*1 NAME(4)
      COMMON MTOT,NP,IA(1)
      NP = 3
      CALL DEFIN(NAME,NA,NR,NC)
      RETURN
      END

C------------------------------------------------------------------ DEFIN
      SUBROUTINE DEFIN(NAME,NA,NR,NC)
C-----DEFINE AND RESERVE STORAGE FOR ARRAY -----

      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC

      CHARACTER*1  NAME(4)
C----DEFIN VARIABLES-------------------------------------------------------
C       NAME - NAME OF ARRAY - 4 LOGICALS MAXIMUM
C       NA - LOCATION OF ARRAY IF IN BLANK COMMON
C       NR - NUMBER OF ROWS
C       NC - NUMBER OF COLUMNS
C       MTOT - END OF DIRECTORY
C       NUMA - NUMBER OF ARRAYS IN DATA BASE
```

```
C       NEXT - NEXT AVAILABLE STORAGE LOCATION
C       IDIR - START OF DIRECTORY IN BLANK COMMON
C       IP   - NUMBER OF LOGICALS CONTAINED IN DATA TYPE
C       LENR - NUMBER OF LOGICALS IN PHYSICAL RECORD
C       NP - TYPE OF DATA
C          = 1 INTEGER DATA
C          = 2 REAL DATA
C          = 3 LOGICAL DATA
C------DIRECTORY DEFINITION FOR CORE OR SEQUENTIAL FILES
C      IDIR(1,N) = NAME OF ARRAY  - INAME (4 CHAR.)
C      IDIR(5,N) = NUMBER OF ROWS    - NR
C      IDIR(6,N) = NUMBER OF COLUMNS - NC
C      IDIR(7,N) = TYPE OF DATA      - NP
C      IDIR(8,N) = INCORE ADDRESS    - NA
C                = -1 IF SEQUENTIAL FILE ON DISK
C                = -2 IF DIRECT ACCESS ON DISK
C      IDIR(9,N) = SIZE OF ARRAY
C      IDIR(10,N) = 0 IF IN CORE STORAGE
C------DIRECTORY DEFINITION FOR DIRECT ACCESS FILES -----
C      IDIR(5,N) = NUMBER OF INTEGERS
C      IDIR(6,N) = NUMBER OF REAL WORDS
C      IDIR(7,N) = NUMBER OF LOGICALS
C      IDIR(8,N) = NUMBER OF LOGICAL RECORDS
C      IDIR(9,N) = LOGICAL RECORD NUMBER
C      IDIR(10,N)  = LUN IF ON LOGICAL UNIT LUN
C-------------------------------------------------------------------------
C-----EVALUATE STORAGE REQUIREMENTS -----
      NSIZE = (NR*NC*IP(NP) -1)/(IP(1)*2)
      NSIZE = NSIZE*2 + 2
      NA = NEXT
      NEXT = NEXT + NSIZE
C-----SET UP NEW DIRECTORY -----
      NUMA = NUMA + 1
      IDIR = IDIR - 10
      I = IDIR
C-----CHECK STORAGE LIMITS -----
      IF(I.GE.NEXT) GO TO 100
      I = NEXT - I + MTOT - 1
      WRITE(NTW,2000) I,MTOT
      WRITE(NOT,2000) I,MTOT
      PAUSE
      STOP
  100 CALL ICON(NAME,IA(I))
      IA(I+4) = NR
      IA(I+5) = NC
      IA(I+6) = NP
      IA(I+7) = NA
      IA(I+8) = NSIZE
      IA(I+9) = 0
 900  RETURN
 2000 FORMAT(
     *' **** ERROR: Insufficient blank COMMON storage.',/,
     *'             Storage required  MTOT =',I7,/,
     *'             Storage available MTOT =',I7)
      END

C-------------------------------------------------------------------DEFDIR
      SUBROUTINE DEFDIR(NAME,NR,NC,ISTR)
C--SUB:DEFDIR - DEFINE DIRECTORY FOR OUT-OF-CORE FILE
C               NAME - NAME OF ARRAY
C               NR   - NUMBER OF ROWS
C               NC   - NUMBER OF COLUMNS
C               ISTR - OUT OF CORE FLAG (=-1)
C-------------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC

      CHARACTER*1 NAME(4)
C-----EVALUATE STORAGE REQUIREMENTS ---------------
      IF(NP.EQ.0) NP = 2
C-----SET UP NEW DIRECTORY -----
      NUMA  = NUMA + 1
      IDIR = IDIR - 10
      I = IDIR
C-----CHECK STORAGE LIMITS -----
      IF(I.GE.NEXT) GO TO 100
      I = NEXT - I + MTOT - 1
      WRITE(NTW,2000) I,MTOT
      WRITE(NOT,2000) I,MTOT
      PAUSE
      STOP
  100 CALL ICON(NAME,IA(I))
      IA(I+4) = NR
      IA(I+5) = NC
      IA(I+6) = NP
      IA(I+7) = ISTR
      IA(I+8) = 0
      IA(I+9) = 0
 900  RETURN
 2000 FORMAT(
    *' **** ERROR: Insufficient blank COMMON storage.',/,
    *'             Storage required  MTOT =',I7,/,
    *'             Storage available MTOT =',I7)
      END

C-----------------------------------------------------------------LOCATE
      SUBROUTINE LOCATE(NAME,NA,NR,NC)
C--SUB:LOCATE - LOCATE ARRAY "NAME" AND RETURN
C              NA - POINTER TO LOCATION IN BLANK COMMON
C              NR - NUMBER OF ROWS
C              NC - NUMBER OF COLUMNS
C-----------------------------------------------------------------------
      COMMON MTOT,NP,IA(1)

      CHARACTER*1 NAME
      DIMENSION NAME(4),INAME(4)
C-----LOCATE AND RETURN PROPERTIES ON ARRAY ---------
      NA = 0
      CALL ICON(NAME,INAME)
      I = IFIND(INAME,0)
      IF(I.EQ.0) GO TO 900
C-----RETURN ARRAY PROPERTIES -----
      NA = IA(I+7)
      NR = IA(I+4)
      NC = IA(I+5)
      NP = IA(I+6)
  900 RETURN
      END

C----------------------------------------------------------------- DELETE
      SUBROUTINE DELETE(NAME)
C--SUB:DELETE - DELETE ARRAY "NAME" FROM DATABASE

      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC
      INCLUDE IOCOM.INC

      CHARACTER*1 NAME
      DIMENSION NAME(4),INAME(4)
C-----DELETE ARRAY FROM STORAGE ------------------
  100 CALL ICON(NAME,INAME)
      I = IFIND(INAME,0)
      IF(I.EQ.0) GO TO 900
C-----CHECK ON STORAGE LOCATION -----
  200 NSIZE = IA(I+8)
C-----SET SIZE OF ARRAY -----
      NEXT = NEXT - NSIZE
      NUMA = NUMA - 1
      NA   = IA(I+7)
C-----CHECK IF OUT OF CORE OR DIRECT ACCESS -----
      IF(NA.GT.0) GO TO 500
      WRITE(NTW,1000) NAME
      WRITE(NOT,1000) NAME
      GO TO 800
  500 IF(NA.EQ.NEXT) GO TO 800
C-----COMPACT STORAGE -----
      II   = NA + NSIZE
      NNXT = NEXT - 1
      DO 700 J=NA,NNXT
      IA(J) = IA(II)
  700 II = II + 1
C-----COMPACT AND UPDATE DIRECTORY -----
  800 NA = I - IDIR
      IDIR = IDIR + 10
      IF(NA.EQ.0) GO TO 900
      NA = NA/10
      DO 860 K=1,NA
      II = I + 9
      DO 850 J=1,10
      IA(II) = IA(II-10)
  850 II = II - 1
      IF(IA(I+7).LE.0) GO TO 860
      IF(IA(I+9).EQ.0) IA(I+7) = IA(I+7) - NSIZE
  860 I = I - 10
C
  900 RETURN
 1000 FORMAT('    -- Name ',4A1,' is being used for an',
     * ' OUT-OF-CORE file.',/)
      END

C-------------------------------------------------------------------ICON
      SUBROUTINE ICON(NAME,INAME)
      CHARACTER*1 NAME(4)
      DIMENSION INAME(4)
C-----CONVERT LOGICALS TO INTEGER DATA ------------
      DO 100 I = 1,4
  100 INAME(I) = ICHAR( NAME(I) )
C
      RETURN
      END

C------------------------------------------------------------------ IFIND
      FUNCTION IFIND(INAME,LUN)
C--FUN:IFIND - FIND
      COMMON MTOT,NP,IA(1)
      INCLUDE ARYCOM.INC

      DIMENSION INAME(4)
C-----FIND ARRAY LOCATION -------------------------
      I = IDIR
      DO 100 N=1,NUMA
      IF(LUN.NE.IA(I+9)) GO TO 100
      IF (INAME(1).NE.IA(I  )) GO TO 100
      IF (INAME(2).NE.IA(I+1)) GO TO 100
      IF (INAME(3).NE.IA(I+2)) GO TO 100
      IF (INAME(4).EQ.IA(I+3)) GO TO 200
  100 I = I + 10
      I = 0
  200 IFIND = I
C
      RETURN
      END

C
C 3.0 MATRIX OPERATION UTILITIES
C

C------------------------------------------------------------------ZEROI
      SUBROUTINE ZEROI(IA,NR,NC)
C--SUB:ZERORI - SET ARRAY IA(NR,NC) TO 0
      DIMENSION IA(NR,NC)
      DO 10 I=1,NR
      DO 10 J=1,NC
      IA(I,J) = 0
   10 CONTINUE
      RETURN
      END

C------------------------------------------------------------------ZEROR
      SUBROUTINE ZEROR(A,NR,NC)
C--SUB:ZEROR - SET ARRAY A(NR,NC) TO 0.0
      REAL*8 A(NR,NC)
      DO 10 I=1,NR
      DO 10 J=1,NC
      A(I,J) = 0.0D0
   10 CONTINUE
      RETURN
      END

C-----------------------------------------------------------------FACTCA
      SUBROUTINE FACTCA(A,NEQ,MBAND,ERR)
C--SUB:FACTCA - FACTORS COMPACT ASYMMETRIC MATRIX
C               FACTORS [A] = [L][U]
C               [L][U] IS WRITTEN OVER [A]
C               [A] MAYBE SYM OR ASYM, POSITIVE DEFINITE
C               [A] HAS SEMI-BANDWIDTH MBAND & IS STORED COMPACTLY
C     FROM: HUEBNER & THORNTON "THE FINITE ELEMENT METHOD FOR ENGRS."
```

```
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE IOCOM.INC

      DIMENSION A(NEQ,2*MBAND-1)
      LOGICAL ERR

      NCOLS = 2*MBAND-1
      KMIN  = MBAND + 1
      DO 50 N=1,NEQ
      IF(A(N,MBAND).EQ.0.0D0) GO TO 60
      IF(A(N,MBAND).EQ.1.0D0) GO TO 20
      C = 1.0D0/A(N,MBAND)
      DO 10 K=KMIN,NCOLS
      IF(A(N,K).EQ.0.0D0) GO TO 10
      A(N,K) = C*A(N,K)
   10 CONTINUE
   20 CONTINUE
      DO 40 L=2,MBAND
      JJ = MBAND - L + 1
      I = N + L - 1
      IF(I.GT.NEQ) GO TO 40
      IF(A(I,JJ).EQ.0.0D0) GO TO 40
      KI = MBAND + 2 - L
      KF = NCOLS + 1 - L
      J = MBAND
      DO 30 K=KI,KF
      J = J + 1
      IF(A(N,J).EQ.0.0D0) GO TO 30
      A(I,K) = A(I,K) - A(I,JJ)*A(N,J)
   30 CONTINUE
   40 CONTINUE
   50 CONTINUE
      RETURN
   60 ERR = .TRUE.
      WRITE(NTW,2000) N
      WRITE(NOT,2000) N
      RETURN
 2000 FORMAT(' **** ERROR: SUB:FACTCA - Equations may be singular.',/,
     +'             Diagonal of equation number ',I5,' is zero.')
      END

C------------------------------------------------------------------SOLVCA
      SUBROUTINE SOLVCA(A,B,NEQ,MBAND,ERR)
C--SUB:SOLVCA -SOLVES COMPACT ASYMMETRIC FACTORED MATRIX
C             SOLVES [L][U](X) = (B)
C             [L][U] IS WRITTEN OVER [A]
C             [L][U]=[A] HAS SEMI-BANDWIDTH MBAND & IS STORED COMPACTLY
C             SOLUTION IS WRITTEN OVER (B)
C     FROM: HUEBNER & THORNTON "THE FINITE ELEMENT METHOD FOR ENGRS."
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)

      INCLUDE IOCOM.INC

      DIMENSION A(NEQ,2*MBAND-1), B(NEQ)
      LOGICAL ERR

      NCOLS = 2*MBAND-1

C--1.0 REDUCTION OF (B)

      DO 30 N=1,NEQ
      IF(A(N,MBAND).EQ.0.0D0) GO TO 60
      IF(A(N,MBAND).EQ.1.0D0) GO TO 10
      B(N) = B(N)/A(N,MBAND)
   10 CONTINUE
      DO 20 L=2,MBAND
      JJ = MBAND - L + 1
      I = N + L -1
      IF(I.GT.NEQ) GO TO 20
      IF(A(I,JJ).EQ.0.0D0) GO TO 20
      B(I) = B(I) - A(I,JJ)*B(N)
   20 CONTINUE
   30 CONTINUE

C--2.0 BACKSUBSTITUTION
```

```
      LL = MBAND + 1
      DO 50 M=1,NEQ
      N = NEQ + 1 - M
      DO 40 L=LL,NCOLS
      IF(A(N,L).EQ.0.0D0) GO TO 40
      K = N + L - MBAND
      B(N) = B(N) - A(N,L)*B(K)
   40 CONTINUE
   50 CONTINUE
      RETURN
   60 ERR = .TRUE.
      WRITE(NTW,2000) N
      RETURN
 2000 FORMAT(' **** ERROR: SUB:SOLVCA - Equations may be singular.',/,
     +'             Diagonal of equation number ',I5,' is zero.')
      END

C------------------------------------------------------------------EIGEN2
      SUBROUTINE EIGEN2(A,T,N,TMX,EP)
C--SUB: EIGEN2 - Unsymmetric Eigen Analysis Routine
C       Based on code from:
C               Wilkinson, J.H. & Reinsch, C., Linear Algebra, Springer-
C               Verlag, 1971
C       Solves eigenproblem for real matrix A(N,N), sym. or unsym., by
C       a sequence of Jacobi-like transformations [T]-1[A][T] where [T]=
C       [T1][T2][T3] ....  Each [Ti] is of the form [Ri][Si] where;
C
C       R:      Rk,k = Rm,m = cos(x)    ;       Rm,k = -Rk,m = sin(x)
C               Ri,i = 1        ;       Ri,j = 0        ; (i,j = k,m)
C       S:      Sk,k = Sm,m = cosh(y)   ;       Sm,k = Sk,m = -sinh(y)
C               Si,i = 1        ;       Si,j = 0        ; (i,j = k,m)
C
C       in which x,y are determined by the elements of [Ai].
C
C       In the limiting matrix real eigenvalues occupy the diagonal while
C       real and imaginary parts of complex eigen values occupy the
C       diagonal and off-diagonal corners of 2x2 blocks centered on diag.
C
C       Array T(N,N) must be provided to receive eigenvectors.
C               TMX=0   : eigenvectors not generated and A(N,N) may be
C                         passed as T(N,N)
C               TMX<0   : generate left, [T]-1, transformations
C               TMX>0   : generate right, [T], transformations
C       Eigenvectors of real eigenvalues occurr as rows (cols) of [T]-1
C       ([T]).  Eigenvectors for a complex eigenvalue pair aj,i 1 iaj,j+1
C       may be formed by tj 1 itj+1 where tj, tj+1 are the corresponding
C       rows (cols) of [T]-1 ([T])
C
C       Iterations are limited to 50 maximum. On exit from the procedure
C       TMX records the number of iterations performed. Failure to
C       converge is indicated by TMX=50 or, if all transformations in
C       one iteration are the identity matrix, by TMX<0.
C
C       The machine dependent variable EP is set to 1E-08 and should be
C       reset for machine precision available.
C-----------------------------------------------------------------------
C-- D I C T I O N A R Y   O F   V A R I A B L E S ---------------------
C
C-----VARIABLE-----------DESCRIPTION------------------------------------
C----INPUT
C     A(N,N)            Array to be analyzed.
C     N                 System size
C     TMS               Control parameter
C-----OUTPUT
C     T(N,N)            Array to receive eigenvectors.
C     TMX               Iteration count/iteration flag
C----LOCAL
C     EP                Precision
C-----------------------------------------------------------------------
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 A(N,N),T(N,N),EP
      INTEGER N,TMX
      LOGICAL MARK, LEFT, RIGHT
C
C--0.0 INITIALIZE CONTROL VARIABLES
C
      IF(EP.LE.0.0D0) EP = 1.0D-8
      EPS = SQRT(EP)
      LEFT = .FALSE.
      RIGHT = .FALSE.
      IF(TMX.LT.0) THEN
        LEFT = .TRUE.
      ELSEIF(TMX.GT.0) THEN
         RIGHT = .TRUE.
      ENDIF
      MARK = .FALSE.
C
C--1.0  INITIALIZE [T] AS IDENTITY MATRIX
C
      IF(TMX.NE.0) THEN
        DO 10 I=1,N
          T(I,I) = 1.0D0
        DO 10 J=I+1,N
          T(I,J) = 0.0D0
          T(J,I) = 0.0D0
   10   CONTINUE
      ENDIF
C
C—2.0 MAIN LOOP
C
      DO 26 IT=1,50
C
C—2.1 IF MARK IS SET
C          TRANSFORMATIONS OF PREVIOUS ITERATION WERE OMITTED
C          PROCEEDURE WILL NOT CONVERGE
C
      IF(MARK) THEN
        TMX = 1-IT
        RETURN
      ENDIF
C
C—2.2 COMPUTE CONVERGENCE CRITERIA
C
      DO 20 I=1,N-1
        AII = A(I,I)
      DO 20 J=I+1,N
        AIJ = A(I,J)
        AJI = A(J,I)
        IF((ABS(AIJ+AJI).GT.EPS).OR.
     +     ((ABS(AIJ-AJI).GT.EPS).AND.(ABS(AII-A(J,J)).GT.EPS))) THEN
          GOTO 21
        ENDIF
   20 CONTINUE
      TMX = IT -1
      RETURN
C
C—2.3 BEGIN NEXT TRANSFORMATION
C
   21 MARK = .TRUE.
      DO 25 K=1,N-1
      DO 25 M=K+1,N
        H = 0.0D0
        G = 0.0D0
        HJ = 0.0D0
        YH = 0.0D0
        DO 22 I=1,N
          AIK = A(I,K)
          AIM = A(I,M)
          TE  = AIK*AIK
          TEE = AIM*AIM
          YH  = YH + TE - TEE
          IF((I.NE.K).AND.(I.NE.M)) THEN
            AKI = A(K,I)
            AMI = A(M,I)
            H   = H + AKI*AMI - AIK*AIM
            TEP = TE + AMI*AMI
            TEM = TEE + AKI*AKI
            G   = G + TEP + TEM
            HJ  = HJ - TEP + TEM
          ENDIF
   22   CONTINUE
        H   = H + H
        D   = A(K,K) - A(M,M)
        AKM = A(K,M)
        AMK = A(M,K)
        C   = AKM + AMK
        E   = AKM - AMK
C
C------ COMPUTE ELEMENTS OF [Ri]
C
        IF(ABS(C).LE.EP) THEN
          CX = 1.0D0
          SX = 0.0D0
        ELSE
          COT2X = D/C
          SIG = SIGN(1.0,COT2X)
          COTX = COT2X + (SIG*SQRT(1.0D0 + COT2X*COT2X))
          SX = SIG/SQRT(1.0D0 + COTX*COTX)
          CX = SX*COTX
        ENDIF

        IF(YH.LT.0.0D0) THEN
          TEM = CX
          CX = SX
          SX = -TEM
        ENDIF

        COS2X = CX*CX - SX*SX
        SIN2X = 2.0D0*SX*CX
        D = D*COS2X + C*SIN2X
        H = H*COS2X - HJ*SIN2X
        DEN = G + 2.0D0*(E*E + D*D)
        TANHY = (E*D - H/2.0D0)/DEN
C
C------ COMPUTE ELEMENTS OF [Si]
C
        IF(ABS(TANHY).LE.EP) THEN
          CHY = 1.0D0
          SHY = 0.0D0
        ELSE
          CHY = 1.0D0/SQRT(1.0D0 - TANHY*TANHY)
          SHY = CHY*TANHY
        ENDIF
C
C------ COMPUTE ELEMENTS OF [Ti] = [Ri][Si]
C
        C1 =  CHY*CX - SHY*SX
        C2 =  CHY*CX + SHY*SX
        S1 =  CHY*SX + SHY*CX
        S2 = -CHY*SX + SHY*CX
C
C------ APPLY TRANSFORMATION IF WARRANTED
C
        IF((ABS(S1).GT.EP).OR.(ABS(S2).GT.EP)) THEN
          MARK = .FALSE.
C-------- TRANSFORMATION ON THE LEFT
          DO 23 I=1,N
            AKI = A(K,I)
            AMI = A(M,I)
            A(K,I) = C1*AKI + S1*AMI
            A(M,I) = S2*AKI + C2*AMI
            IF(LEFT) THEN
              TKI = T(K,I)
              TMI = T(M,I)
              T(K,I) = C1*TKI + S1*TMI
              T(M,I) = S2*TKI + C2*TMI
            ENDIF
   23     CONTINUE
C-------- TRANSFORMATION ON THE RIGHT
          DO 24 I=1,N
            AIK = A(I,K)
            AIM = A(I,M)
            A(I,K) = C2*AIK - S2*AIM
            A(I,M) = -S1*AIK + C1*AIM
            IF(RIGHT) THEN
              TIK = T(I,K)
              TIM = T(I,M)
              T(I,K) = C2*TIK - S2*TIM
              T(I,M) = -S1*TIK + C1*TIM
            ENDIF
   24     CONTINUE
        ENDIF
   25 CONTINUE
   26 CONTINUE
      TMX = 50

      RETURN
      END
```

Include Files:

```
C-----------------------------------------------------------------------
C CALSAPX  A R R A Y   M A N A G E M E N T                  $ARYCOM.INC
C-----------------------------------------------------------------------
      COMMON /ARYCOM/ NUMA,NEXT,IDIR,IP(3)

C----VARIABLE--------DESCRIPTION----------------------------------------
C       MTOT          SIZE OF BLANK COMMON VECTOR IA
C       NP            CURRENT DATA TYPE: 1=INTEGER; 2=REAL; 3=CHAR.
C       IA(MTOT)      BLANK COMMON VECTOR
C       NUMA          NUMBER OF ARRAYS IN BLANK COMMON DATA BASE
C       NEXT          NEXT AVAILABLE STORAGE LOCATION IN BLANK COMMON
C       IDIR          START OF DIRECTORY IN BLANK COMMON
C       IP(3)         NUMBER OF BYTES IN INTEGER, REAL, CHARACTER DATA
C-----------------------------------------------------------------------


C-----------------------------------------------------------------------
C CALSAPX  I/O  F I L E   M A N A G E M E N T               $IOCOM.INC
C-----------------------------------------------------------------------
      INTEGER LFNAME
      LOGICAL ECHO,EOD,EOC
      CHARACTER*1  FNAME*12, EXT*3, MODE*5
      COMMON /IOCOM1/NTR,NTW,NCMD,NIN,NOT,ND1,ND2,ND3,ND4,
     +LFNAME,ECHO,EOD,EOC
      COMMON /IOCOM2/ MODE,EXT,FNAME
C----VARIABLE--------DESCRIPTION----------------------------------------
C    /IOCOM/
C      NTR             LOGICAL UNIT NUMBER FOR TERMINAL-READ (KEYBOARD)
C      NTW             LOGICAL UNIT NUMBER FOR TERMINA-WRITE (SCREEN)
C      NCMD            LOGICAL UNIT NUMBER FOR COMMAND/DATA INPUT
C      NIN             LOGICAL UNIT NUMBER FOR INPUT DATA ASCII FILE
C      NOT             LOGICAL UNIT NUMBER FOR OUTPUT DATA ASCII FILE
C      ND1 thru ND4    LOGICAL UNIT NUMBERS FOR GENERAL USE
C      FNAME*12        RESULTS OUTPUT FILE NAME
C      LFNAME          LENGTH OF FILENAME WITH TRAILING BLANKS REMOVED
C      EXT*3           RESULTS OUTPUT FILE EXTENSION
C      MODE            COMMAND MODE: 'INTER'=INTERACTIVE, 'BATCH'=BATCH
C      ECHO            WHEN .TRUE. ECHO RESULTS OUTPUT TO NTW (SCREEN)
C      EOD             END-OF-DATA LOGICAL
C      EOC             END-OF-COMMAND LOGICAL
C-----------------------------------------------------------------------


C-----------------------------------------------------------------------
C CALSAPX  F R E E - F I E L D   I N P U T                  $FRECOM.INC
C-----------------------------------------------------------------------
      CHARACTER LINE*1, LLINE*160
      COMMON /CLINE1/ LINE(160)
      COMMON /CLINE2/ II,JJ
      EQUIVALENCE (LLINE,LINE(1))
      SAVE /CLINE1/,/CLINE2/

C----VARIABLE--------DESCRIPTION----------------------------------------
C       LINE(160)      COMMAND LINE BUFFER
C       II             END-OF-DATA IN LINE BUFFER
C       JJ             END-OF-INFORMATION IN LINE BUFFER
C                      INFORMATION = DATA : COMMENTS
C-----------------------------------------------------------------------


C-----------------------------------------------------------------------
C CALSAPX  C O M M A N D   M A N A G E M E N T              $CMDCOM.INC
C-----------------------------------------------------------------------
      CHARACTER*1 NCMND,M1,M2,M3,M4,M5,M6,M7,NNCMND*8
      COMMON /CMND/ NCMND(8),M1(4),M2(4),M3(4),M4(4),M5(4),M6(4),M7(4)
      EQUIVALENCE (NNCMND,NCMND(1))

C----VARIABLE--------DESCRIPTION----------------------------------------
C       NCMND(8)*1     CURRENT COMMAND
C       NNCMND*8       CURRENT COMMAND
C       M1(4) to M7(4) CURRENT ARRAY NAMES
C-----------------------------------------------------------------------


C-----------------------------------------------------------------------
C CONTAN  C O M M O N   S T O R A G E                      $CNTCOM86.INC
C-----------------------------------------------------------------------
      REAL*8 EP
      COMMON /CNTCOM/NFNOD,NFEQN,NFBAN,NFELM,EP,
     +MPV,MPF,MPC,MPG,MPKEQ,MPWE
C----VARIABLE--------DESCRIPTION----------------------------------------
C       ERR            DO-WHILE TERMINATOR FLAG
C       NFNOD          NUMBER OF FLOW SYSTEM NODES
C       NFEQN          NUMBER OF FLOW SYSTEM EQUATIONS
C                      = NFNOD (CURRENT VERSION)
C       NFBAN          (HALF) BANDWIDTH OF FLOW SYSTEM EQUATIONS
C       NFELM          NUMBER OF FLOW ELEMENTS
C       EP             MACHINE PRECISION
C
C    P O I N T E R S   T O   B L A N K   C O M M O N   L O C A T I O N S
C-------POINTER--ARRAY--------------------------------------------------
C
C       MPV      V(NSNOD)         : VOLUMETRIC MASSES
C       MPF    F(NFEQN,2*MSBAN-1) : FLOW MATRIX (UNSYMMETRIC)
C       MPC      C(NFEQN)         : CONTAMINANT CONCENTRATION
C       MPG      G(NFEQN)         : CONTAMINANT GENERATION
C       MPKEQ  KEQ(NFNOD)         : SYSTEM EQUATION NUMBERS
C                                 :   0 = UNDEFINED
C                                 : NEG = CONCENTRATION PRESCRIBED DOF
C                                 : POS = GENERATION PRESCRIBED DOF
C       MPWE    WE(NFELM)         : ELEMENT MASS FLOW RATES
C
C-----------------------------------------------------------------------
```